Synthesis of Hierarchical Systems

Benjamin Aminof[a], Fabio Mogavero[b], and Aniello Murano[b⋆]

[a]Hebrew University, Jerusalem 91904, Israel.
[b]Università degli Studi di Napoli "Federico II", 80126 Napoli, Italy.
benj@cs.huji.ac.il {mogavero, murano}@na.infn.it

Abstract In automated synthesis, given a specification, we automatically create a system that is guaranteed to satisfy the specification. In the classical temporal synthesis algorithms, one usually creates a "flat" system "from scratch". However, real-life software and hardware systems are usually created using pre-existing libraries of reusable components, and are not "flat" since repeated subsystems are described only once.
In this work we describe an algorithm for the synthesis of a hierarchical system from a library of hierarchical components, which follows the "bottom-up" approach to system design. Our algorithm works by synthesizing in many rounds, when at each round the system designer provides the specification of the currently desired module, which is then automatically synthesized using the initial library and the previously constructed modules. To ensure that the synthesized module actually takes advantage of the available high-level modules, we guide the algorithm by enforcing certain modularity criteria.
We show that the synthesis of a hierarchical system from a library of hierarchical components is EXPTIME-complete for μ-calculus, and 2EXPTIME-complete for LTL, both in the cases of complete and incomplete information. Thus, in all cases, it is not harder than the classical synthesis problem (of synthesizing flat systems "from scratch"), even though the synthesized hierarchical system may be exponentially smaller than a flat one.

1 Introduction

Synthesis is the automated construction of a system from its specification. The basic idea is simple and appealing: instead of developing a system and verifying that it is correct w.r.t. its specification, we use instead an automated procedure that, given a specification, constructs a system that is correct by construction. The first formulation of synthesis goes back to Church [7]; the modern approach to this problem was initiated by Pnueli and Rosner who introduced linear temporal logic (LTL) synthesis [23], later extended to handle branching-time specifications, such as μ-calculus [10].

The Pnueli and Rosner idea can be summarized as follows. Given sets Σ_I and Σ_O of inputs and outputs, respectively (usually, $\Sigma_I = 2^I$ and $\Sigma_O = 2^O$, where I is a set of input signals and O is a set of output signals), we can view a system as a strategy $P : \Sigma_I^* \to \Sigma_O$ that maps a finite sequence of sets of input signals into a set of output signals. When P interacts with an environment that generates infinite input sequences, it associates with each input sequence an infinite computation over $\Sigma_I \cup \Sigma_O$. Though the system P is deterministic, it induces a computation tree. The branches of the tree

⋆ Partially supported by ESF GAMES grant 4112 and Vigevani Project Prize 2010-2011.

correspond to external nondeterminism, caused by different possible inputs. Thus, the tree has a fixed branching degree $|\Sigma_I|$, and it embodies all the possible inputs (and hence also computations) of P. When we synthesize P from an LTL specification φ, we require φ to hold in all the paths of P's computation tree. However, in order to impose possibility requirements on P, we have to use a branching-time logic like μ-calculus. Given a branching specification φ over $\Sigma_I \cup \Sigma_O$, realizability of φ is the problem of determining whether there exists a system P whose computation tree satisfies φ. Correct synthesis of φ then amounts to constructing such a P.

In spite of the rich theory developed for system synthesis in the last two decades, little of this theory has been reduced to practice. In fact, the main approaches to tackle synthesis in practice are either to use heuristics (e.g., [13]) or to restrict to simple specifications (e.g., [22]). Some people argue that this is because the synthesis problem is very expensive compared to model-checking [16]. There is, however, something misleading in this perception: while the complexity of synthesis is given with respect to the specification only, the complexity of model-checking is given also with respect to a program, which can be very large. A common thread in almost all of the works concerning synthesis is the assumption that the system is to be built "from scratch". Obviously, real-world systems are rarely constructed this way, but rather by utilizing many preexisting reusable components, i.e., a library. Using standard preexisting components is sometimes unavoidable (for example, access to hardware resources is usually under the control of the operating system, which must be "reused"), and many times has other benefits (apart from saving time and effort, which may seem to be less of a problem in a setting of automatic - as opposed to manual - synthesis), such as maintaining a common code base, and abstracting away low level details that are already handled by the preexisting components. Another reason that may also account, at least partially, for the limited use of synthesis in practice, is the fact that synthesized systems are usually monolithic and look very unnatural from the system designer's point of view. Indeed, in classical temporal synthesis algorithms, one usually creates a "flat" system, i.e., a system in which sub-systems may be repeated many times. On the contrary, real-life software and hardware systems are hierarchical (or even recursive) and repeated subsystems (such as sub-routines) are described only once. While hierarchical systems may be exponentially more succinct than flat ones, it has been shown that the cost of solving questions about them (like model-checking) are in many cases not exponentially higher [5, 6, 12]. Hierarchical systems can also be seen as a special case of recursive systems [2, 3], where the nesting of calls to sub-systems is bounded. However, having no bound on the nesting of calls gives rise to infinite-state systems, and this results in a higher complexity.

In this work we provide a uniform algorithm, for different temporal logics, for the synthesis of hierarchical systems (or, equivalently, *transducers*) from a library of hierarchical systems, which mimics the "bottom-up" approach to system design, where one builds a system by iteratively constructing new modules based on previously constructed ones[1]. More specifically, we start the synthesis process by providing the algorithm with an initial library $\mathcal{L}_0$ of available hierarchical components (as well as atomic ones). We then proceed by synthesizing in rounds. At each round i, the system designer provides a specification formula φ_i of the currently desired hierarchical component,

[1] While for systems built from scratch, a top-down approach may be argued to be more suitable, we find the bottom-up approach to be more natural when synthesizing from a library.

which is then automatically synthesized using the currently available components as possible sub-components. Once a new component is synthesized, it is added to the library to be used by subsequent iterations. We show that while hierarchical systems may be exponentially smaller than flat ones, the problem of synthesizing a hierarchical system from a library of existing hierarchical systems is EXPTIME-complete for μ-calculus, and 2EXPTIME-complete for LTL. Thus, this problem is not harder than the classical synthesis problem of flat systems "from scratch". Furthermore, we show that this is true also in the case where the synthesized system has incomplete information about the environment's input.

Observe that it is easily conceivable that if the initial library $\mathcal{L}_0$ contains enough atomic components then the synthesis algorithm may use them exclusively, essentially producing a flat system. We thus have to direct the single-round synthesis algorithm in such a way that it produces modular and not flat results. The question of what makes a design more or less modular is very difficult to answer, and has received many (and often widely different) answers throughout the years (see [21], for a survey). We claim that some very natural modularity criteria are regular, and show how any criterion that can be checked by a parity tree automaton can be easily incorporated into our automata based synthesis algorithm.

Related work The issues of specification and correctness of modularly designed systems have received a fair attention in the formal verification literature. Examples of important work on this subject are [8, 17, 26, 27]. On the other hand, the problem of automatic synthesis from reusable components, which we study here, has received much less attention. The closest to our work is Lustig and Vardi's work on LTL synthesis from libraries of (flat) transducers [18]. The technically most difficult part of our work is an algorithm for performing the synthesis step of a single round of the multiple-rounds algorithm. To this end, we use an automata-theoretic approach. However, unlike the classical approach of [23], we build an automaton whose input is not a computation tree, but rather a system description in the form of a *connectivity tree* (inspired by the "control-flow" trees of [18]), which describes how to connect library components in a way that satisfies the specification formula. Taken by itself, our single-round algorithm extends the "control-flow" synthesis work from [18] in four directions. **(i)** We consider not only LTL specifications but also the modal μ-calculus. Hence, unlike [18], where co-Büchi tree automata were used, we have to use the more expressive parity tree automata. Unfortunately, this is not simply a matter of changing the acceptance condition. Indeed, in order to obtain an optimal upper bound, a widely different approach, which makes use of the machinery developed in [6] is needed. **(ii)** We need to be able to handle libraries of hierarchical transducers, whereas in [18] only libraries of flat transducers are considered. **(iii)** A synthesized transducer has no top-level exits (since it must be able to run on all possible input words), and thus, its ability to serve as a sub-transducer of another transducer (in future iterations of the multiple-rounds algorithm) is severely limited (it is like a function that never returns to its caller). We therefore need to address the problem of synthesizing exits for such transducers. **(iv)** As discussed above, we incorporate into the algorithm the enforcing of modularity criteria.

Recently, an extension of [18] appeared in [19], where the problem of *Nested-Words Temporal Logic* (NWTL) synthesis from recursive component libraries has been investigated. NWTL extends LTL with special operators that allow one to handle "call and return" computations [1] and it is used in [19] to describe how the components have to be connected in the synthesis problem. We recall that in our framework the logic

does not drive (at least not explicitly) the way the components have to be connected. Moreover, the approach used in [19] cannot be applied directly to the branching framework we consider in this paper, as we recall that already the satisfiability problem for μ-calculus with "call and return" is undecidable even for very restricted cases [4].

2 Alternating Tree Automata

Let $\mathcal{D}$ be a set. A *$\mathcal{D}$-tree* is a prefix-closed subset $T \subseteq \mathcal{D}^*$ such that if $x \cdot c \in T$, where $x \in \mathcal{D}^*$ and $c \in \mathcal{D}$, then also $x \in T$. The *complete $\mathcal{D}$-tree* is the tree $\mathcal{D}^*$. The elements of T are called *nodes*, and the empty word ε is the *root* of T. Given a word $x = y \cdot d$, with $y \in \mathcal{D}^*$ and $d \in \mathcal{D}$, we define $last(x)$ to be d. For $x \in T$, the nodes $x \cdot d \in T$, where $d \in \mathcal{D}$, are the *sons* of x. A *leaf* is a node with no sons. A *path* of T is a set $\pi \subseteq T$ such that $\varepsilon \in T$ and, for every $x \in \pi$, either x is a leaf or there is a unique $d \in \mathcal{D}$ such that $x \cdot d \in \pi$. For an alphabet Σ, a *Σ-labeled $\mathcal{D}$-tree* is a pair $\langle T, V \rangle$ where $T \subseteq \mathcal{D}^*$ is a $\mathcal{D}$-tree and $V : T \to \Sigma$ maps each node of T to a letter in Σ.

Alternating tree automata are a generalization of nondeterministic tree automata [20] (see [16], for more details). Intuitively, while a nondeterministic tree automaton that visits a node of the input tree sends exactly one copy of itself to each of the sons of the node, an alternating automaton can send several copies of itself to the same son.

An (asymmetric) *Alternating Parity Tree Automaton* (*APT*) is a tuple $\mathcal{A} = \langle \Sigma, \mathcal{D}, Q, q_0, \delta, F \rangle$, where Σ, $\mathcal{D}$, and Q are non-empty finite sets of *input letters*, *directions*, and *states*, respectively; $q_0 \in Q$ is an *initial state*, F is a *parity acceptance condition* to be defined later, and $\delta : Q \times \Sigma \mapsto \mathcal{B}^+(\mathcal{D} \times Q)$ is an *alternating transition function*, which maps a state and an input letter to a positive boolean combination of elements in $\mathcal{D} \times Q$. Given a set $\mathcal{S} \subseteq \mathcal{D} \times Q$ and a formula $\theta \in \mathcal{B}^+(\mathcal{D} \times Q)$, we say that $\mathcal{S}$ satisfies θ (denoted by $\mathcal{S} \models \theta$) if assigning **true** to elements in $\mathcal{S}$ and **false** to elements in $(\mathcal{D} \times Q) \setminus \mathcal{S}$, makes θ true. A *run* of an APT $\mathcal{A}$ on a Σ-labeled $\mathcal{D}$-tree $\mathcal{T} = \langle T, V \rangle$ is a $(T \times Q)$-labeled $\mathbb{N}$-tree $\langle T_r, r \rangle$, where $\mathbb{N}$ is the set of non-negative integers, such that *(i)* $r(\varepsilon) = (\varepsilon, q_0)$ and *(ii)* for all $y \in T_r$, with $r(y) = (x, q)$, there exists a set $\mathcal{S} \subseteq \mathcal{D} \times Q$, such that $\mathcal{S} \models \delta(q, V(x))$, and there is one son y' of y, with $r(y') = (x \cdot d, q')$, for every $(d, q') \in \mathcal{S}$. Given a node x of a run $\langle T_r, r \rangle$, with $r(x) = (z, q) \in T \times Q$, we define $last(r(y)) = (last(z), q)$. An alternating parity automaton $\mathcal{A}$ is nondeterministic (denoted NPT), iff when its transition relation is rewritten in disjunctive normal form each disjunct contains at most one element of $\{d\} \times Q$, for every $d \in \mathcal{D}$. An automaton is universal (denoted UPT) if all the formulas that appear in its transition relation are conjunctions of atoms in $\mathcal{D} \times Q$.

A *symmetric* alternating parity tree automaton with ε-moves (*SAPT*) [14] does not distinguish between the different sons of a node, and can send copies of itself only in a universal or an existential manner. Formally, an SAPT is a tuple $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F \rangle$, where Σ is a finite input alphabet; Q is a finite set of states, partitioned into universal ($Q^\wedge$), existential ($Q^\vee$), ε-and ($Q^{(\varepsilon,\wedge)}$), and ε-or ($Q^{(\varepsilon,\vee)}$) states (we also write $Q^{\vee,\wedge} = Q^\vee \cup Q^\wedge$, and $Q^\varepsilon = Q^{(\varepsilon,\vee)} \cup Q^{(\varepsilon,\wedge)}$); $q_0 \in Q$ is an initial state; $\delta : Q \times \Sigma \to (Q \cup 2^Q)$ is a transition function such that for all $\sigma \in \Sigma$, we have that $\delta(q, \sigma) \in Q$ for $q \in Q^{\vee,\wedge}$, and $\delta(q, \sigma) \in 2^Q$ for $q \in Q^\varepsilon$; and F is a parity acceptance condition, to be defined later. We assume that Q contains in addition two special states `ff` and `tt`, called *rejecting sink* and *accepting sink*, respectively, such that $\forall a \in \Sigma : \delta(\mathtt{tt}, a) = \mathtt{tt}, \delta(\mathtt{ff}, a) = \mathtt{ff}$. The classification of `ff` and `tt` is arbitrary. Transitions from states in Q^ε launch copies of $\mathcal{A}$ that stay on the same input node as before the transition, while transitions from states in $Q^{\vee,\wedge}$ launch copies that advance to sons of the current node (note that for an SAPT

the set $\mathcal{D}$ of directions of the input trees plays no role in the definition of a run). When a symmetric alternating tree automaton $\mathcal{A}$ runs on an input tree it starts with a copy in state q_0 whose reading head points to the root of the tree. It then follows δ in order to send further copies. For example, if a copy of $\mathcal{A}$ that is in state $q \in Q^{(\varepsilon,\vee)}$ is reading a node x labeled σ, and $\delta(q,\sigma) = \{q_1, q_2\}$, then this copy proceeds either to state q_1 or to state q_2, and its reading head stays in x. As another example, if $q \in Q^{\wedge}$ and $\delta(q,\sigma) = q_1$, then $\mathcal{A}$ sends a copy in state q_1 to every son of x. Note that different copies of $\mathcal{A}$ may have their reading head pointing to the same node of the input tree. Formally, a *run* of $\mathcal{A}$ on a Σ-labeled $\mathcal{D}$-tree $\langle T, V\rangle$ is a $(T \times Q)$-labeled $\mathbb{N}$-tree $\langle T_r, r\rangle$. A node in T_r labeled by (x,q) describes a copy of $\mathcal{A}$ in state q that reads the node x of T. A run has to satisfy $r(\varepsilon) = (\varepsilon, q_0)$ and, for all $y \in T_r$ with $r(y) = (x,q)$, the following hold:

- If $q \in Q^{\wedge}$ (resp. $q \in Q^{\vee}$) and $\delta(q, V(x)) = p$, then for each son (resp. for exactly one son) $x \cdot d$ of x, there is a node $y \cdot i \in T_r$ with $r(y \cdot i) = (x \cdot d, p)$.
- If $q \in Q^{(\varepsilon,\wedge)}$ (resp. $q \in Q^{(\varepsilon,\vee)}$) and $\delta(q, V(x)) = \{p_0, ..., p_k\}$, then for all $i \in \{0..k\}$ (resp. for one $i \in \{0..k\}$) the node $y \cdot i \in T_r$, and $r(y \cdot i) = (x, p_i)$;

A *parity condition* is given by means of a coloring function on the set of states. Formally, a *parity condition* is a function $F : Q \to C$, where $C = \{C_{\min}, ..., C_{\max}\} \subset \mathbb{N}$ is a set of colors. The size $|C|$ of C is called the *index* of the automaton. For an SAPT, we also assume that the special state `tt` is given an even color, and `ff` is given an odd color. For an infinite path $\pi \subseteq T_r$ of a run $\langle T_r, r\rangle$, let $maxC(\pi)$ be the maximal color that appears infinitely often along π. Similarly, for a finite path π, we define $maxC(\pi)$ to be the maximal color that appears at least once in π. An infinite path $\pi \subseteq T_r$ satisfies the acceptance condition F iff $maxC(\pi)$ is even. A run $\langle T_r, r\rangle$ is accepting iff all its infinite paths satisfy F. The automaton $\mathcal{A}$ accepts an input tree $\langle T, V\rangle$ if there is an accepting run of $\mathcal{A}$ on $\langle T, V\rangle$. The language of $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of Σ-labeled $\mathcal{D}$-trees accepted by $\mathcal{A}$. We say that an automaton $\mathcal{A}$ is nonempty iff $\mathcal{L}(\mathcal{A}) \neq \emptyset$.

A wide range of branching-time temporal logics can be translated to alternating tree automata (details can be found in [16]). In particular:

Theorem 1. *[11, 16] Given a temporal-logic formula φ, it is possible to construct a SAPT $\mathcal{A}_\varphi$ such that $\mathcal{L}(\mathcal{A}_\varphi)$ is exactly the set of trees satisfying φ. Moreover,*

- *if φ is a μ-calculus formula, then $\mathcal{A}_\varphi$ is an alternating parity automaton with $O(|\varphi|)$ states and index $O(|\varphi|)$; and*
- *if φ is an* LTL *formula, then $\mathcal{A}_\varphi$ is a universal parity automaton with $2^{O(|\varphi|)}$ states, and index 2.*

3 Hierarchical Transducers

In this section, we introduce *hierarchical transducers* (alternatively, *hierarchical Moore machines*), which are a generalization of classical transducers in which repeated substructures (technically, *sub-transducers*) are specified only once. Technically, some of the states in a hierarchical transducer are *boxes*, in which inner hierarchical transducers are nested. Formally, a hierarchical transducer is a tuple $\mathcal{K} = \langle \Sigma_I, \Sigma_O, \langle \mathcal{K}_1, ..., \mathcal{K}_n\rangle\rangle$, where Σ_I and Σ_O are respectively non-empty sets of input and output letters, and for every $1 \leq i \leq n$, the sub-transducer $\mathcal{K}_i = \langle W_i, \mathcal{B}_i, in_i, Exit_i, \tau_i, \delta_i, \Lambda_i\rangle$ has the following elements.

- W_i is a finite set of *states*. $in_i \in W_i$ is an *initial state*[2], and $Exit_i \subseteq W_i$ is a set of *exit-states*. States in $W_i \setminus Exit_i$ are called *internal states*.
- A finite set $\mathcal{B}_i$ of *boxes*. We assume that $W_1,...,W_n,\mathcal{B}_1,...,\mathcal{B}_n$ are pairwise disjoint.
- An *indexing function* $\tau_i : \mathcal{B}_i \to \{i+1,...,n\}$ that maps each box of the i-th sub-transducer to a sub-transducer with an index greater than i. If $\tau_i(b) = j$ we say that b *refers* to $\mathcal{K}_j$.
- A *transition function* $\delta_i : (\bigcup_{b\in\mathcal{B}_i}(\{b\} \times Exit_{\tau_i(b)}) \cup (W_i \setminus Exit_i)) \times \Sigma_I \to W_i \cup \mathcal{B}_i$. Thus, when the transducer is at an internal state $u \in (W_i \setminus Exit_i)$, or at an exit e of a box b, and it reads an input letter $\sigma \in \Sigma_I$, it moves either to a state $s \in W_i$, or to a box $b' \in \mathcal{B}_i$. A move to a box b' implicitly leads to the unique initial state of the sub-transducer that b' refers to.
- A *labeling function* $\Lambda_i : W_i \to \Sigma_O$ that maps states to output letters.

The sub-transducer $\mathcal{K}_1$ is called the *top-level* sub-transducer of $\mathcal{K}$. Thus, for example, the top-level boxes of $\mathcal{K}$ are the elements of $\mathcal{B}_1$, etc. We also call in_1 the initial state of $\mathcal{K}$, and $Exit_1$ the *exits* of $\mathcal{K}$. For technical convenience we sometimes refer to functions (like the transitions and labeling functions) as relations, and in particular, we consider $\emptyset$ to be a function with an empty domain. Note that the fact that boxes can refer only to sub-transducers of a greater index implies that the nesting depth of transducers is finite. In contrast, in the *recursive* setting such a restriction does not exist. Also note that moves from an exit $e \in Exit_i$ of a sub-transducer $\mathcal{K}_i$ are not specified by the transition function δ_i of $\mathcal{K}_i$, but rather by the transition functions of the sub-transducers that contain boxes that refer to $\mathcal{K}_i$. The exits of $\mathcal{K}$ allow us to use it as a sub-transducer of another hierarchical transducer. When we say that a hierarchical transducer $\mathcal{K} = \langle \Sigma_I, \Sigma_O, \langle \mathcal{K}_1,..., \mathcal{K}_n \rangle \rangle$ is a sub-transducer of another hierarchical transducer $\mathcal{K}' = \langle \Sigma_I, \Sigma_O, \langle \mathcal{K}'_1,..., \mathcal{K}'_{n'} \rangle \rangle$, we mean that $\{\mathcal{K}_1,..., \mathcal{K}_n\} \subseteq \{\mathcal{K}'_2,..., \mathcal{K}'_{n'}\}$. The size $|\mathcal{K}_i|$ of a sub-transducer $\mathcal{K}_i$ is the sum $|W_i| + |\mathcal{B}_i| + |\delta_i|$. The size $|\mathcal{K}|$ of $\mathcal{K}$ is the sum of the sizes of its sub-transducers. We sometimes abuse notation and refer to the *hierarchical transducer* $\mathcal{K}_i$ which is formally the hierarchical transducer $\langle \Sigma_I, \Sigma_O, \langle \mathcal{K}_i, \mathcal{K}_{i+1},..., \mathcal{K}_n \rangle \rangle$ obtained by taking $\mathcal{K}_i$ to be the top-level sub-transducer.

Flat transducers A sub-transducer without boxes is *flat*. A hierarchical transducer $\mathcal{K} = \langle \Sigma_I, \Sigma_O, \langle W, \emptyset, in, Exit, \emptyset, \delta, \Lambda \rangle \rangle$ with a single (hence flat) sub-transducer is flat, and we denote it using the shorter notation $\mathcal{K} = \langle \Sigma_I, \Sigma_O, \langle W, in, Exit, \delta, \Lambda \rangle \rangle$. Each hierarchical transducer $\mathcal{K}$ can be transformed into an equivalent flat transducer $\mathcal{K}^f = \langle \Sigma_I, \Sigma_O, \langle W^f, in_1, Exit_1, \delta^f, \Lambda^f \rangle \rangle$ (called its *flat expansion*) by recursively substituting each box by a copy of the sub-transducer it refers to. Since different boxes can refer to the same sub-transducer, states may appear in different contexts. In order to obtain unique names for states in the flat expansion, we prefix each copy of a sub-transducer's state by the sequence of boxes through which it is reached. Thus, a state $(b_0,...,b_k,w)$ of $\mathcal{K}^f$ is a vector whose last component w is a state in $\cup_{i=1}^{n} W_i$, and the remaining components $(b_0,...,b_k)$ are boxes that describe its context. The labeling of a state $(b_0,...,b_k,w)$ is determined by its last component w. For simplicity, we refer to vectors of length one as elements (that is, w, rather than (w)).[3] Formally, given a hierarchical transducer

[2] We assume a single entry for each sub-transducer. Multiple entries can be handled by duplicating sub-transducers.

[3] A helpful way to think about this is using a stack — the boxes $b_0,...,b_k$ are pushed into the stack whenever a sub-transducer is called, and are popped in the corresponding exit.

$\mathcal{K} = \langle \Sigma_I, \Sigma_O, \langle \mathcal{K}_1, ..., \mathcal{K}_n \rangle \rangle$, for each sub-transducer $\mathcal{K}_i = \langle W_i, \mathcal{B}_i, in_i, Exit_i, \tau_i, \delta_i, \Lambda_i \rangle$ we inductively define its flat expansion $\mathcal{K}_i^{\mathrm{f}} = \langle W_i^{\mathrm{f}}, in_i, Exit_i, \delta_i^{\mathrm{f}}, \Lambda_i^{\mathrm{f}} \rangle$ as follows.

- The set of states $W_i^{\mathrm{f}} \subseteq W_i \cup (B_i \times (\bigcup_{j=i+1}^{n} W_j^{\mathrm{f}}))$ is defined as follows: (i) if w is a state of W_i then w belongs to W_i^{f}; and (ii) if b is a box of $\mathcal{K}_i$ with $\tau_i(b) = j$, and the tuple $(u_1,...,u_h)$ is a state in W_j^{f}, then $(b, u_1,...,u_h)$ belongs to W_i^{f}.
- The transition function δ_i^{f} is defined as follows: (i) If $\delta_i(u,\sigma) = v$, where $u \in W_i$, or $u = (b,e)$ with $b \in \mathcal{B}_i$ and $e \in Exit_{\tau_i(b)}$, then if v is a state, we have that $\delta_i^{\mathrm{f}}(u,\sigma) = v$; and if v is a box, we have that $\delta_i^{\mathrm{f}}(u,\sigma) = (v, in_{\tau_i(v)})$. Note that $(v, in_{\tau_i(v)})$ is indeed a state of W_i^{f} by the second item in the definition of states above; and (ii) if b is a box of $\mathcal{K}_i$, and $\delta_{\tau_i(b)}^{\mathrm{f}}((u_1,...,u_h),\sigma) = (v_1,...,v_{h'})$ is a transition of $\mathcal{K}_{\tau_i(b)}^{\mathrm{f}}$, then $\delta_i^{\mathrm{f}}((b,u_1,...,u_h),\sigma) = (b,v_1,...,v_{h'})$ is a transition of $\mathcal{K}_i^{\mathrm{f}}$.
- Finally, if $u \in W_i$ then $\Lambda_i^{\mathrm{f}}(u) = \Lambda_i(u)$; and if $u \in W_i^{\mathrm{f}}$ is of the form $u = (b,u_1,...,u_h)$, where $b \in \mathcal{B}_i$, then $\Lambda_i(u) = \Lambda_{\tau_i(b)}^{\mathrm{f}}(u_1,...,u_h)$.

The transducer $\langle \Sigma_I, \Sigma_O, \langle \mathcal{K}_1^{\mathrm{f}} \rangle \rangle$ is the required flat expansion $\mathcal{K}^{\mathrm{f}}$ of $\mathcal{K}$. An *atomic transducer* is a flat transducer made up of a single node that serves as both an entry and an exit. For each letter $\varsigma \in \Sigma_O$ there is an atomic transducer $K_\varsigma = \langle \{p\}, p, \{p\}, \emptyset, \{(p,\varsigma)\} \rangle$ whose single state p is labeled by ς.

Run of a hierarchical transducer Consider a hierarchical transducer $\mathcal{K}$ with $Exit_1 = \emptyset$ that interacts with its environment. At point j in time, the environment provides $\mathcal{K}$ with an input $\sigma_j \in \Sigma_I$, and in response $\mathcal{K}$ moves to a new state, according to its transition relation, and outputs the label of that state. The result of this infinite interaction is a computation of $\mathcal{K}$, called the *trace* of the run of $\mathcal{K}$ on the word $\sigma_1 \cdot \sigma_2 \cdots$. In the case that $Exit_1 \neq \emptyset$, the interaction comes to a halt whenever $\mathcal{K}$ reaches an exit $e \in Exit_1$, since top-level exits have no outgoing transitions. Formally, a *run* of a hierarchical transducer $\mathcal{K}$ is defined by means of its flat expansion $\mathcal{K}^{\mathrm{f}}$. Given a finite input word $v = \sigma_1 \cdots \sigma_m \in \Sigma_I^*$, a *run* (*computation*) of $\mathcal{K}$ on v is a sequence of states $r = r_0 \cdots r_m \in (W^{\mathrm{f}})^*$ such that $r_0 = in_1$, and $r_j = \delta^{\mathrm{f}}(r_{j-1}, \sigma_j)$, for all $0 < j \leq m$. Note that since $\mathcal{K}$ is deterministic it has at most one run on every word, and that if $Exit_1 \neq \emptyset$ then $\mathcal{K}$ may not have a run on some words. The *trace* of the run of $\mathcal{K}$ on v is the word of inputs and outputs $\mathrm{trc}(\mathcal{K}, v) = (\Lambda^{\mathrm{f}}(r_1), \sigma_1) \cdots (\Lambda^{\mathrm{f}}(r_m), \sigma_m) \in (\Sigma_O \times \Sigma_I)^*$. The notions of traces and runs are extended to infinite words in the natural way.

The computations of $\mathcal{K}$ can be described by a *computation tree* whose branches correspond to the runs of $\mathcal{K}$ on all possible inputs, and whose labeling gives the traces of these runs. Note that the root of the tree corresponds to the empty word ε, and its labeling is not part of any trace. However, if we look at the computation tree of $\mathcal{K}$ as a sub-tree of a computation tree of a transducer $\mathcal{K}'$ of which $\mathcal{K}$ is a sub-transducer, then the labeling of the root of the computation tree of $\mathcal{K}$ is meaningful, and it corresponds to the last element in the trace of the run of $\mathcal{K}'$ leading to the initial state of $\mathcal{K}$. Formally, given $\sigma \in \Sigma_I$, the computation tree $\mathcal{T}_{\mathcal{K},\sigma} = \langle T_{\mathcal{K},\sigma}, V_{\mathcal{K},\sigma} \rangle$, is a $(\Sigma_O \times \Sigma_I)$-labeled $(W^{\mathrm{f}} \times \Sigma_I)$-tree, where: *(i)* the root ε is labeled by $(\Lambda^{\mathrm{f}}(in_1), \sigma)$; *(ii)* a node $y = (r_1,\sigma_1) \cdots (r_m,\sigma_m) \in (W^{\mathrm{f}} \times \Sigma_I)^+$ is in $T_{\mathcal{K},\sigma}$ iff $in_1 \cdot r_1 \cdots r_m$ is the run of $\mathcal{K}$ on $v = \sigma_1 \cdots \sigma_m$, and its label is $V_{\mathcal{K},\sigma}(y) = (\Lambda^{\mathrm{f}}(r_m), \sigma_m)$. Thus, for a node y, the labels of the nodes on the path from the root (excluding the root) to y are exactly $\mathrm{trc}(\mathcal{K}, v)$. Observe that the leaves of $\mathcal{T}_{\mathcal{K},\sigma}$ correspond to pairs (e, σ'), where $e \in Exit_1$ and $\sigma' \in \Sigma_I$.

However, if $Exit_1 = \emptyset$, then the tree has no leaves, and it represents the runs of $\mathcal{K}$ over all words in Σ_I^*. We sometimes consider a leaner computation tree $\mathcal{T}_\mathcal{K} = \langle T_\mathcal{K}, V_\mathcal{K} \rangle$ that is a Σ_O-labeled Σ_I-tree, where a node $y \in \Sigma_I^+$ is in $T_\mathcal{K}$ iff there is a run r of $\mathcal{K}$ on y. The label of such a node is $V_\mathcal{K}(y) = \Lambda^{\mathrm{f}}(last(r)))$ and the label of the root is $\Lambda^{\mathrm{f}}(in_1)$. Observe that for every $\sigma \in \Sigma_I$, the tree $\mathcal{T}_\mathcal{K}$ can be obtained from $\mathcal{T}_{\mathcal{K},\sigma}$ by simply deleting the first component of the directions of $\mathcal{T}_{\mathcal{K},\sigma}$, and the second component of the labels of $\mathcal{T}_{\mathcal{K},\sigma}$.

Recall that the labeling of the root of a computation tree of $\mathcal{K}$ is not part of any trace (when it is not a sub-tree of another tree). Hence, in the definition below, we arbitrarily fix some letter $\rho \in \Sigma_I$. Given a temporal logic formula φ, over the atomic propositions AP where $2^{AP} = \Sigma_O \times \Sigma_I$, we have the following:

Definition 1. *A hierarchical transducer $\mathcal{K} = \langle \Sigma_I, \Sigma_O, \langle \mathcal{K}_1, ..., \mathcal{K}_u \rangle \rangle$, with $Exit_1 = \emptyset$, satisfies a formula φ (written $\mathcal{K} \models \varphi$), iff the tree $\mathcal{T}_{\mathcal{K},\rho}$ satisfies φ.*

Observe that given φ, finding a flat transducer $\mathcal{K}$ such that $\mathcal{K} \models \varphi$ is the classic synthesis problem studied (for LTL formulas) in [23].

A *library* $\mathcal{L}$ is a finite set of hierarchical transducers with the same input and output alphabets. Formally, $\mathcal{L} = \{\mathcal{K}^1, ..., \mathcal{K}^\lambda\}$, and for every $1 \leq i \leq \lambda$, we have that $\mathcal{K}^i = \langle \Sigma_I, \Sigma_O, \langle \mathcal{K}^i_1, ..., \mathcal{K}^i_{n_i} \rangle \rangle$. Note that a transducer in the library can be a sub-transducer of another one, or share common sub-transducers with it. The set of transducers in $\mathcal{L}$ that have no top-level exits is denoted by $\mathcal{L}^{=\emptyset} = \{\mathcal{K}^i \in \mathcal{L} : Exit^i_1 = \emptyset\}$, and its complement is $\mathcal{L}^{\neq\emptyset} = \mathcal{L} \setminus \mathcal{L}^{=\emptyset}$.

4 Hierarchical Synthesis

In this section, we describe our synthesis algorithm. We start by providing the algorithm with an initial library $\mathcal{L}_0$ of hierarchical transducers. A good starting point is to include in $\mathcal{L}_0$ all the atomic transducers, as well as any other relevant hierarchical transducers, for example from a standard library. We then proceed by synthesizing in rounds. At each round $i \geq 0$, the system designer provides a specification formula φ_i of the currently desired hierarchical transducer $\mathcal{K}^i$, which is then automatically synthesized using the transducers in $\mathcal{L}_{i-1}$ as possible sub-transducers. Once a new transducer is synthesized it is added to the library, for use in subsequent rounds. Technically, the hierarchical transducer synthesized in the last round as the output of the algorithm.

Input: An initial library $\mathcal{L}_0$, and a list of specification formulas $\varphi_1, ..., \varphi_m$
Output: A hierarchical transducer satisfying φ_m
for $i = 1$ **to** m **do**
  synthesize $\mathcal{K}^i$ satisfying φ_i using the transducers in $\mathcal{L}_{i-1}$ as sub-transducers
  $\mathcal{L}_i \leftarrow \mathcal{L}_{i-1} \cup \{\mathcal{K}^i\}$
end
return $\mathcal{K}^m$

Algorithm 1: Hierarchical Synthesis Algorithm

The main challenge in implementing the above hierarchical synthesis algorithm is of course coming up with an algorithm for performing the synthesis step of a single round. As noted in Section 1, a transducer that was synthesized in a previous round has no top-level exits, which severely limits its ability to serve as a sub-transducer of another transducer. Our single-round algorithm must therefore address the problem of

synthesizing exits for such transducers. In Section 4.1, we give our core algorithm for single-round synthesis of a hierarchical transducer from a given library of hierarchical transducers. In Section 4.2, we address the problem of enforcing modularity, and add some more information regarding the synthesis of exits. Finally, in Section 4.3, we address the problem of synthesis with imperfect information.

4.1 Hierarchical Synthesis from a Library

We now formally present the problem of hierarchical synthesis from a library (that may have transducers without top-level exits) of a single temporal logic formula. Given a transducer $\mathcal{K} = \langle \Sigma_I, \Sigma_O, \langle \mathcal{K}_1, ..., \mathcal{K}_n \rangle \rangle \in \mathcal{L}^{=\emptyset}$, where $\mathcal{K}_1 = \langle W_1, \mathcal{B}_1, in_1, \emptyset, \tau_1, \delta_1, \Lambda_1 \rangle$, and a set $E \subseteq W_1$, the transducer $\mathcal{K}^E$ is obtained from $\mathcal{K}$ by setting E to be the set of top-level exits, and removing all the outgoing edges from states in E. Formally, $\mathcal{K}^E = \langle \Sigma_I, \Sigma_O, \langle \langle W_1, \mathcal{B}_1, in_1, E, \tau_1, \delta'_1, \Lambda_1 \rangle, \mathcal{K}_2, ..., \mathcal{K}_n \rangle \rangle$, where the transition relation δ'_1 is the restriction of δ_1 to sources in $W_1 \setminus E$. For convenience, given a transducer $\mathcal{K} \in \mathcal{L}^{\neq\emptyset}$ we sometimes refer to it as $\mathcal{K}^{Exit_1}$. For every $\mathcal{K} \in \mathcal{L}$, we assume some fixed ordering on the top-level states of $\mathcal{K}$, and given a set $E \subseteq W_1$, and a state $e \in E$, we denote by $idx(e,E)$ the relative position of e in E, according to this ordering. Given a library $\mathcal{L}$, and an upper bound $el \in \mathbb{N}$ on the number of allowed top-level exits, we let $\mathcal{L}^{el} = \mathcal{L}^{\neq\emptyset} \cup \{ \mathcal{K}^E : \mathcal{K} \in \mathcal{L}^{=\emptyset} \wedge |E| \leq el \}$. The higher the number el, the more exits the synthesis algorithm is allowed to synthesize, and the longer it may take to run. As we show later, el should be at most polynomial[4] in the size of φ. In general, we assume that el is never smaller than the number of exits in any sub-transducer of any hierarchical transducer in $\mathcal{L}$. Hence, for every $\mathcal{K}^E \in \mathcal{L}^{el}$ and every $e \in E$, we have that $1 \leq idx(e,E) \leq el$.

Definition 2. *Given a library $\mathcal{L}$ and a bound $el \in \mathbb{N}$, we say that:*

- *A hierarchical transducer $\mathcal{K} = \langle \Sigma_I, \Sigma_O, \langle \mathcal{K}_1, ... \mathcal{K}_n \rangle \rangle$ is* $\langle \mathcal{L}, el \rangle$-composed *if (i) for every $2 \leq i \leq n$, we have that $\mathcal{K}_i \in \mathcal{L}^{el}$; (ii) if $w \in W_1$ is a top-level state, then the atomic transducer $K_{\Lambda_1(w)}$ is in $\mathcal{L}$.*
- *A formula φ is* $\langle \mathcal{L}, el \rangle$-realizable *iff there is an $\langle \mathcal{L}, el \rangle$-composed hierarchical transducer $\mathcal{K}$ that satisfies φ. The $\langle \mathcal{L}, el \rangle$-synthesis problem is to find such a $\mathcal{K}$.*

Intuitively, an $\langle \mathcal{L}, el \rangle$-composed hierarchical transducer $\mathcal{K}$ is built by synthesizing its top-level sub-transducer $\mathcal{K}_1$, which specifies how to connect boxes that refer to transducers from $\mathcal{L}^{el}$. To eliminate an unnecessary level of indirection, boxes that refer to atomic transducers are replaced by regular states. Note that this also solves the technical problem that, by definition, the initial state in_1 cannot be a box. This is also the reason why we assume from now on that every library has at least one atomic transducer. Note that for each transducer $\mathcal{K}' \in \mathcal{L}^{=\emptyset}$ we can have as many as $\Omega(|\mathcal{K}'|)^{el}$ copies of $\mathcal{K}'$ in $\mathcal{L}^{el}$, each with a different set of exit states. In Section 4.2 we show how, when we synthesize $\mathcal{K}$, we can limit the number of such copies that $\mathcal{K}$ uses to any desired value (usually one per $\mathcal{K}'$).

[4] In practical terms, the exits of a sub-module represent its set of possible return values. Since finite state modules are usually not expected to have return values over large domains (such as the set of integers), we believe that our polynomial bound for el is not too restrictive.

Connectivity trees In the classical automata-theoretic approach to synthesis [23], synthesis is reduced to finding a regular tree that is a witness to the non-emptiness of a suitable tree automaton. Such a regular tree is simply an infinite labeled tree where its structure represents all the possible infinite input sequences and the labeling represents mapping of inputs to outputs. Since every transducer induces an execution tree, and every regular execution tree can be implemented by a transducer, the classical synthesis problem for a temporal logic specification φ reduces to first constructing a tree automaton whose language is the set of the above described trees that also satisfy φ and then to check the emptiness of this automaton. Here, we also reduce synthesis to the non-emptiness problem of a tree automaton. However, unlike the classical approach, we build an automaton whose input is not a computation tree, but rather a system description in the form of a *connectivity tree* (inspired by the "control-flow" trees of [18]), which describes how to connect library components in a way that satisfies the specification formula. Specifically, given a library $\mathcal{L} = \{\mathcal{K}^1,...,\mathcal{K}^\lambda\}$ and a bound $el \in \mathbb{N}$, connectivity trees represent hierarchical transducers that are $\langle \mathcal{L}, el\rangle$-composed, in the sense that every regular $\langle \mathcal{L}, el\rangle$-composed hierarchical transducer induces a connectivity tree, and vice versa. Formally, a *connectivity tree* $\mathcal{T} = \langle T, V\rangle$ for $\mathcal{L}$ and el, is an $\mathcal{L}^{el}$-labeled complete $(\{1,...,el\} \times \Sigma_I)$-tree, where the root is labeled by an atomic transducer.

Intuitively, a node x with $V(x) = \mathcal{K}^E$ represents a top-level state q if $\mathcal{K}^E$ is an atomic transducer, and otherwise it represents a top-level box b that refers to $\mathcal{K}^E$. The label of a son $x \cdot (idx(e,E), \sigma)$ specifies the destination of the transition from the exit e of b (or from a state q, if $\mathcal{K}^E$ is atomic — in which case it has a single exit) when reading σ. Sons $x \cdot (i,e)$, for which $i > |E|$, are ignored. Thus, a path $\pi = (i_0,\sigma_0) \cdot (i_1,\sigma_1) \cdots$ in a connectivity tree $\mathcal{T}$ is called *meaningful*, iff for every $j > 0$, we have that i_j is not larger than the number of top-level exits of $V(i_{j-1},\sigma_{j-1})$.

A connectivity tree $\mathcal{T} = \langle T, V\rangle$ is *regular* if there is a flat transducer $\mathcal{M} = \langle \{1,...,el\} \times \Sigma_I, \mathcal{L}^{el}, \langle M, m_0, \emptyset, \delta^{\mathcal{T}}, \Lambda^{\mathcal{T}}\rangle\rangle$, such that $\mathcal{T}$ is equal to the (lean) computation tree $\mathcal{T}_{\mathcal{M}}$. A regular connectivity tree induces an $\langle \mathcal{L}, el\rangle$-composed hierarchical transducer $\mathcal{K}$, whose top-level sub-transducer $\mathcal{K}_1$ is basically a replica of $\mathcal{M}$ (see the Appendix for the reverse transformation). Every node $m \in M$ becomes a state of $\mathcal{K}_1$ if $\Lambda^{\mathcal{T}}(m)$ is an atomic-transducer and, otherwise, it becomes a box of $\mathcal{K}_1$ which refers to the top-level sub-transducer of $\Lambda^{\mathcal{T}}(m)$. The destination of a transition from an exit e of a box m, with $\Lambda^{\mathcal{T}}(m) = \mathcal{K}^E$, when reading a letter $\sigma \in \Sigma_I$, is given by $\delta^{\mathcal{T}}(m, (idx(e,E), \sigma))$. If m is a state, then $\Lambda^{\mathcal{T}}(m)$ is an atomic transducer with a single exit and thus, the destination of a transition from m when reading a letter $\sigma \in \Sigma_I$, is given by $\delta^{\mathcal{T}}(m, (1,\sigma))$. For a box b of $\mathcal{K}_1$, let $\Lambda^{\mathcal{T}}(b) = \langle \Sigma_I, \Sigma_O, \langle \mathcal{K}_{(b,1)},... \mathcal{K}_{(b,n_b)}\rangle\rangle$, and denote by $sub(b) = \{\mathcal{K}_{(b,1)},... \mathcal{K}_{(b,n_b)}\}$ the set of sub-transducers of $\Lambda^{\mathcal{T}}(b)$, and by $E(b)$ the set of top-level exits of $\Lambda^{\mathcal{T}}(b)$. Formally, $\mathcal{K} = \langle \Sigma_I, \Sigma_O, \langle \mathcal{K}_1,..., \mathcal{K}_n\rangle\rangle$, where $\mathcal{K}_1 = \langle W_1, \mathcal{B}_1, m_0, \tau_1, \delta_1, \Lambda_1\rangle$, and:

- $W_1 = \{w \in M : \exists \varsigma \in \Sigma_O \text{ s.t. } \Lambda^{\mathcal{T}}(w) = K_\varsigma\}$. Note that since the root of a connectivity tree is labeled by an atomic transducer then $m_0 \in W_1$.
- $\mathcal{B}_1 = M \setminus W_1$.
- The sub-transducers $\{\mathcal{K}_2,..., \mathcal{K}_n\} = \bigcup_{\{b \in B_1\}} sub(b)$.
- For $b \in \mathcal{B}_1$, we have that $\tau_1(b) = i$, where i is such that $\mathcal{K}_i = \mathcal{K}_{(b,1)}$.
- For $w \in W_1$, and $\sigma \in \Sigma_I$, we have that $\delta_1(w,\sigma) = \delta^{\mathcal{T}}(w, (1,\sigma))$.
- For $b \in \mathcal{B}_1$, we have that $\delta_1((b,e),\sigma) = \delta^{\mathcal{T}}(b, (idx(e,E(b)),\sigma))$, for every $e \in E(b)$ and $\sigma \in \Sigma_I$.

– Finally, for $w \in W_1$ we have that $\Lambda_1(w) = \varsigma$, where ς is such that $\Lambda^{\mathcal{T}}(w) = K_\varsigma$.

From synthesis to automata emptiness Given a library $\mathcal{L} = \{\mathcal{K}^1,...,\mathcal{K}^\lambda\}$, a bound $el \in \mathbb{N}$, and a temporal logic formula φ, our aim is to build an APT $\mathcal{A}^{\mathcal{T}}_\varphi$ such that $\mathcal{A}^{\mathcal{T}}_\varphi$ accepts a regular connectivity tree $\mathcal{T} = \langle T, V\rangle$ iff it induces a hierarchical transducer $\mathcal{K}$ such that $\mathcal{K} \models \varphi$. Recall that by Definition 1 and Theorem 1, $\mathcal{K} \models \varphi$ iff $\mathcal{T}_{\mathcal{K},\rho}$ is accepted by the SAPT $\mathcal{A}_\varphi$. The basic idea is thus to have $\mathcal{A}^{\mathcal{T}}_\varphi$ simulate all possible runs of $\mathcal{A}_\varphi$ on $\mathcal{T}_{\mathcal{K},\rho}$. Unfortunately, since $\mathcal{A}^{\mathcal{T}}_\varphi$ has as its input not the tree $\mathcal{T}_{\mathcal{K},\rho}$, but the connectivity tree $\mathcal{T}$, this is not a trivial task. In order to see how we can solve this problem, we first have to make the following observation.

Let $\mathcal{T} = \langle T, V\rangle$ be a regular connectivity tree, and let $\mathcal{K}$ be the hierarchical transducer that it induces. Consider a node $u \in T_{\mathcal{K},\rho}$ with $last(u) = ((b, in_{\tau_1(b)}), \sigma)$, where b is some top-level box, or state[5], of $\mathcal{K}$ that refers to some transducer $\mathcal{K}^E$ (note that the root of $T_{\mathcal{K},\rho}$ is such a node). Observe that the sub-tree $\mathcal{T}^u$, rooted at u, represents the traces of computations of $\mathcal{K}$ that start from the initial state of $\mathcal{K}^E$, in the context of the box b. The sub-tree $prune(\mathcal{T}^u)$, obtained by pruning every path in $\mathcal{T}^u$ at the first node $\hat{u}$, with $last(\hat{u}) = ((b,e), \hat{\sigma})$ for some $e \in E$ and $\hat{\sigma} \in \Sigma_I$, represents the portions of these traces that stay inside $\mathcal{K}^E$. Note that $prune(\mathcal{T}^u)$ is essentially independent of the context b in which $\mathcal{K}^E$ appears, and is isomorphic to the tree $\mathcal{T}_{\mathcal{K}^E,\sigma}$ (the isomorphism being to simply drop the component b from every letter in the name of every node in $prune(\mathcal{T}^u)$). Moreover, every son v (in $T_{\mathcal{K},\rho}$), of such a leaf $\hat{u}$ of $prune(\mathcal{T}^u)$, is of the same form as u. I.e., $last(v) = ((b', in_{\tau_1(b')}), \sigma')$, where $b' = \delta_1((b,e), \sigma')$ is a top-level box (or state) of $\mathcal{K}$. It follows that $T_{\mathcal{K},\rho}$ is isomorphic to a concatenation of sub-trees of the form $\mathcal{T}_{\mathcal{K}^E,\sigma}$, where the transition from a leaf of one such sub-tree to the root of another is specified by the transition relation δ_1, and is thus given explicitly by the connectivity tree $\mathcal{T}$.

The last observation is the key to how $\mathcal{A}^{\mathcal{T}}_\varphi$ can simulate, while reading $\mathcal{T}$, all the possible runs of $\mathcal{A}_\varphi$ on $\mathcal{T}_{\mathcal{K},\rho}$. The general idea is as follows. Consider a node u of $T_{\mathcal{K},\rho}$ such that $prune(\mathcal{T}^u)$ is isomorphic to $\mathcal{T}_{\mathcal{K}^E,\sigma}$. A copy of $\mathcal{A}^{\mathcal{T}}_\varphi$ that reads a node y of $\mathcal{T}$ labeled by $\mathcal{K}^E$ can easily simulate, without consuming any input, all the portions of the runs of any copy of $\mathcal{A}_\varphi$ that start by reading u and remain inside $prune(\mathcal{T}^u)$. This simulation can be done by simply constructing $\mathcal{T}_{\mathcal{K}^E,\sigma}$ on the fly and running $\mathcal{A}_\varphi$ on it. For every simulated copy of $\mathcal{A}_\varphi$ that reaches a leaf $\hat{u}$ of $prune(\mathcal{T}^u)$, (recall that $last(\hat{u})$ is of the form $((b,e), \hat{\sigma})$), the automaton $\mathcal{A}^{\mathcal{T}}_\varphi$ sends copies of itself to the sons of y in the connectivity tree, in order to continue the simulation on the different sub-trees rooted at sons of $\hat{u}$ in $T_{\mathcal{K},\rho}$. The simulation of a copy of $\mathcal{A}_\varphi$ that proceeds to a son $v = \hat{u} \cdot ((b', in_{\tau_1(b')}), \sigma')$, where b' is a top-level box (or state) of $\mathcal{K}$, is handled by a copy of $\mathcal{A}^{\mathcal{T}}_\varphi$ that is sent to the son $z = y \cdot (idx(e,E), \sigma')$.

Our construction of $\mathcal{A}^{\mathcal{T}}_\varphi$ implements the above idea, with one main modification. In order to obtain optimal complexity in successive rounds of Algorithm 1, it is important to keep the size of $\mathcal{A}^{\mathcal{T}}_\varphi$ independent of the size of the transducers in the library. Unfortunately, simulating the runs of $\mathcal{A}_\varphi$ on $\mathcal{T}_{\mathcal{K}^E,\sigma}$ on the fly would require an embedding of $\mathcal{K}^E$ inside $\mathcal{A}^{\mathcal{T}}_\varphi$. Recall, however, that no input is consumed by $\mathcal{A}^{\mathcal{T}}_\varphi$ while running such

[5] Here we think of top-level states of $\mathcal{K}$ as boxes that refer to atomic transducers.

a simulation. Hence, we can perform these simulations offline instead, in the process of building the transition relation of $\mathcal{A}^{\mathcal{T}}_{\varphi}$. Obviously, this requires a way of summarizing the possibly infinite number of runs of $\mathcal{A}_{\varphi}$ on $\mathcal{T}_{\mathcal{K}^E,\sigma}$, which we do by employing the concept of summary functions from [6].

First, we define an ordering $\succeq$ on colors by letting $c \succeq c'$ when c is better, from the point of view of acceptance by $\mathcal{A}_{\varphi}$, than c'. Formally, $c \succeq c'$ if the following holds: if c' is even then c is even and $c \geq c'$; and if c' is odd then either c is even, or c is also odd and $c \leq c'$. We denote by $\min^{\succeq}$ the operation of taking the minimal color, according to $\succeq$, of a finite set of colors. Let $\mathcal{A}_{\varphi} = \langle \Sigma_O \times \Sigma_I, Q_{\varphi}, q^0_{\varphi}, \delta_{\varphi}, F_{\varphi} \rangle$, let $\mathcal{A}^q_{\varphi}$ be the automaton $\mathcal{A}_{\varphi}$ using $q \in Q$ as an initial state, and let C be the set of colors used in the acceptance condition F_{φ}. Consider a run $\langle T_r, r \rangle$ of $\mathcal{A}^q_{\varphi}$ on $\mathcal{T}_{\mathcal{K}^E,\sigma}$. Note that if $z \in T_r$ is a leaf, then $last(r(z)) = ((e,\sigma'),q)$, where $q \in Q^{\vee,\wedge}_{\varphi}$ (i.e., q is not an ε-state), and $e \in E$. We define a function $g_r : E \times \Sigma_I \times Q^{\vee,\wedge}_{\varphi} \to C \cup \{\dashv\}$, called the *summary function* of $\langle T_r, r \rangle$, which summarizes this run. Given $h \in E \times \Sigma_I \times Q^{\vee,\wedge}_{\varphi}$, if there is no leaf $z \in T_r$, such that $last(r(z)) = h$, then $g_r(h) = \dashv$; otherwise, $g_r(h) = c$, where c is the maximal color encountered by the copy of $\mathcal{A}_{\varphi}$ which made the least progress towards satisfying the acceptance condition, among all copies that reach a leaf $z \in T_r$ with $last(r(z)) = h$. Formally, let $paths(r,h)$ be the set of all the paths in $\langle T_r, r \rangle$ that end in a leaf $z \in T_r$, with $last(r(z)) = h$. Then, $g_r(h) = \dashv$ if $paths(r,h) = \emptyset$ and, otherwise, $g_r(h) = \min^{\succeq}\{maxC(\pi) : \pi \in paths(r,h)\}$.

Let $Sf(\mathcal{K}^E, \sigma, q)$ be the set of summary functions of the runs of $\mathcal{A}^q_{\varphi}$ on $\mathcal{T}_{\mathcal{K}^E,\sigma}$. If $\mathcal{T}_{\mathcal{K}^E,\sigma}$ has no leaves, then $Sf(\mathcal{K}^E, \sigma, q)$ contains only the empty summary function ε. For $g \in Sf(\mathcal{K}^E, \sigma, q)$, let $g^{\neq\dashv} = \{h \in E \times \Sigma_I \times Q^{\vee,\wedge}_{\varphi} : g(h) \neq \dashv\}$. Based on the ordering $\succeq$ we defined for colors, we can define a partial order $\succeq$ on $Sf(\mathcal{K}^E, \sigma, q)$, by letting $g \succeq g'$ if for every $h \in (E \times \Sigma_I \times Q^{\vee,\wedge}_{\varphi})$ the following holds: $g(h) = \dashv$, or $g(h) \neq \dashv \neq g'(h)$ and $g(h) \succeq g'(h)$. Observe that if r and r' are two non-rejecting runs, and $g_r \succ g_{r'}$, then extending r to an accepting run on a tree that extends $\mathcal{T}_{\mathcal{K}^E,\sigma}$ is always not harder than extending r' - either because $\mathcal{A}_{\varphi}$ has less copies at the leaves of r, or because these copies encountered better maximal colors. Given a summary function g, we say that a run $\langle T_r, r \rangle$ *achieves* g if $g_r \succeq g$; we say that g is *feasible* if there is a run $\langle T_r, r \rangle$ that achieves it; and we say that g is *relevant* if it can be achieved by a memoryless[6] run that is not rejecting (i.e., by a run that has no infinite path that does not satisfy the acceptance condition of $\mathcal{A}_{\varphi}$). We denote by $Rel(\mathcal{K}^E, \sigma, q) \subseteq Sf(\mathcal{K}^E, \sigma, q)$ the set of relevant summary functions.

We are now ready to give a formal definition of the automaton $\mathcal{A}^{\mathcal{T}}_{\varphi}$. Given a library $\mathcal{L} = \{\mathcal{K}^1, ..., \mathcal{K}^{\lambda}\}$, a bound $el \in \mathbb{N}$, and a temporal-logic formula φ, let $\mathcal{A}_{\varphi} = \langle \Sigma_O \times \Sigma_I, Q_{\varphi}, q^0_{\varphi}, \delta_{\varphi}, F_{\varphi} \rangle$, let $C = \{C_{\min}, ..., C_{\max}\}$ be the colors in the acceptance condition of $\mathcal{A}_{\varphi}$, and for $\mathcal{K}^E \in \mathcal{L}^{el}$, let Λ^E be the labeling function of the top-level sub-transducer of $\mathcal{K}^E$. The automaton $\mathcal{A}^{\mathcal{T}}_{\varphi} = \langle \mathcal{L}^{el}, (\{1, ..., el\} \times \Sigma_I), (\Sigma_I \times Q^{\vee,\wedge}_{\varphi} \times C) \cup \{q_0\}, q_0, \delta, \alpha \rangle$, has the following elements:

- For every $\mathcal{K}^E \in \mathcal{L}^{el}$ we have that $\delta(q_0, \mathcal{K}^E) = \delta((\rho, q^0_{\varphi}, C_{min}), \mathcal{K}^E)$ if $\mathcal{K}^E$ is an atomic transducer and, otherwise, $\delta(q_0, \mathcal{K}^E) = \mathbf{false}$.

[6] A run of an automaton $\mathcal{A}$ is memoryless if two copies of $\mathcal{A}$ that are in the same state, and read the same input node, behave in the same way on the rest of the input.

- For every $(\sigma,q,c) \in \Sigma_I \times Q_\varphi^{\vee,\wedge} \times C$, and every $\mathcal{K}^E \in \mathcal{L}^{el}$, we have $\delta((\sigma,q,c),\mathcal{K}^E) = \bigvee_{g \in Rel(\mathcal{K}^E,\sigma,q)} \bigwedge_{(e,\hat{\sigma},\hat{q}) \in g^{\neq\dashv}} \bigoplus_{\sigma' \in \Sigma_I} ((idx(e,E),\sigma'),(\sigma',\delta_\varphi(\hat{q},(\Lambda^E(e),\hat{\sigma})),g(e,\hat{\sigma},\hat{q})))$, where $\bigoplus = \bigwedge$ if $\hat{q} \in Q_\varphi^\wedge$, and $\bigoplus = \bigvee$ if $\hat{q} \in Q_\varphi^\vee$.
- $\alpha(q_0) = C_{min}$; and $\alpha((\sigma,q,c)) = c$, for every $(\sigma,q,c) \in \Sigma_I \times Q_\varphi^{\vee,\wedge} \times C$.

The construction above implies the following lemma:

Lemma 1. *$\mathcal{A}_\varphi^\mathcal{T}$ accepts a regular connectivity tree $\mathcal{T} = \langle T,V\rangle$ iff $\mathcal{T}$ induces a hierarchical transducer $\mathcal{K}$, such that $\mathcal{T}_{\mathcal{K},\rho}$ is accepted by $\mathcal{A}_\varphi$.*

Proof (sketch). Intuitively, $\mathcal{A}_\varphi^\mathcal{T}$ first checks that the root of its input tree $\mathcal{T}$ is labeled by an atomic proposition (and is thus a connectivity tree), and then proceeds to simulate all the runs of $\mathcal{A}_\varphi$ on $\mathcal{T}_{\mathcal{K},\rho}$. A copy of $\mathcal{A}_\varphi^\mathcal{T}$ at a state (σ,q,c), that reads a node y of $\mathcal{T}$ labeled by $\mathcal{K}^E$, considers all the non-rejecting runs of $\mathcal{A}_\varphi^q$ on $\mathcal{T}_{\mathcal{K}^E,\sigma}$, by looking at the set $Rel(\mathcal{K}^E,\sigma,q)$ of summary functions for these runs. It then sends copies of $\mathcal{A}_\varphi^\mathcal{T}$ to the sons of y to continue the simulation of copies of $\mathcal{A}_\varphi$ that reach the leaves of $\mathcal{T}_{\mathcal{K}^E,\sigma}$.

The logic behind the definition of $\delta((\sigma,q,c),\mathcal{K}^E)$ is as follows. Since every summary function $g \in Rel(\mathcal{K}^E,\sigma,q)$ summarizes at least one non-rejecting run, and it is enough that one such run can be extended to an accepting run of $\mathcal{A}_\varphi$ on the remainder of $\mathcal{T}_{\mathcal{K},\rho}$, we have a disjunction on all $g \in Rel(\mathcal{K}^E,\sigma,q)$. Every $(e,\hat{\sigma},\hat{q}) \in g^{\neq\dashv}$ represents one or more copies of $\mathcal{A}_\varphi$ at state $\hat{q}$ that are reading a leaf $\hat{u}$ of $\mathcal{T}_{\mathcal{K}^E,\sigma}$ with $last(\hat{u}) = (e,\hat{\sigma})$, and all these copies must accept their remainders of $\mathcal{T}_{\mathcal{K},\rho}$. Hence, we have a conjunction over all $(e,\hat{\sigma},\hat{q}) \in g^{\neq\dashv}$.

A copy of $\mathcal{A}_\varphi$ that starts at the root of $\mathcal{T}_{\mathcal{K}^E,\sigma}$ may give rise to many copies that reach a leaf $\hat{u}$ of $\mathcal{T}_{\mathcal{K}^E,\sigma}$ with $last(\hat{u}) = (e,\hat{\sigma})$, but we only need to consider the copy which made the least progress towards satisfying the acceptance condition, as captured by $g(e,\hat{\sigma},\hat{q})$. To continue the simulation of such a copy on its remainder of $\mathcal{T}_{\mathcal{K},\rho}$, we send to a son $y \cdot (idx(e,E),\sigma')$ of y in the connectivity tree, whose label specifies where $\mathcal{K}$ should go to from the exit e when reading σ', a copy of $\mathcal{A}_\varphi^\mathcal{T}$ as follows. Recall that the leaf $\hat{u}$ corresponds to a node u of $\mathcal{T}_{\mathcal{K},\rho}$ such that $last(u) = ((b,e),\hat{\sigma})$ and b is a top-level box of $\mathcal{K}$ that refers to $\mathcal{K}^E$. Also recall that every node in $\mathcal{T}_{\mathcal{K},\rho}$ has one son for every letter $\sigma' \in \Sigma_I$. Hence, a copy of $\mathcal{A}_\varphi$ that is at state $\hat{q}$ and is reading u, sends one copy in state $q' = \delta_\varphi(\hat{q},(\Lambda^E(e),\hat{\sigma}))$ to each son of u, if $\hat{q} \in Q_\varphi^\wedge$; and only one such copy, to one of the sons of u, if $\hat{q} \in Q_\varphi^\vee$. This explains why $\bigoplus$ is a conjunction in the first case, and is a disjunction in the second. Finally, a copy of $\mathcal{A}_\varphi^\mathcal{T}$ that is sent to direction $(idx(e,E),\sigma')$ carries with it the color $g(e,\hat{\sigma},\hat{q})$, which is needed in order to define the acceptance condition. The color assigned to q_0 is of course arbitrary.

The core of the proof uses a game based approach. Recall that the game-based approach to model checking a flat system $\mathcal{S}$ with respect to a branching-time temporal logic specification φ, reduces the model-checking problem to solving a game (called the *membership game* of $\mathcal{S}$ and A_φ) obtained by taking the product of $\mathcal{S}$ with the alternating tree automaton $\mathcal{A}_\varphi$ [16]. In [6], this approach was extended to hierarchical structures, and it was shown there that given a hierarchical structure $\mathcal{S}$ and an SAPT $\mathcal{A}$, one can construct a hierarchical membership game $\mathcal{G}_{\mathcal{S},\mathcal{A}}$ such that Player 0 wins $\mathcal{G}_{\mathcal{S},\mathcal{A}}$ iff the tree obtained by unwinding $\mathcal{S}$ is accepted by $\mathcal{A}$. In particular, when $\mathcal{A}$ accepts exactly all the tree models of a branching-time formula φ, the above holds iff $\mathcal{S}$ satisfies φ.

Furthermore, it is shown in [6] that one can *simplify* the hierarchical membership game $\mathcal{G}_{\mathcal{S},\mathcal{A}}$, by replacing boxes of the top-level arena with gadgets that are built using Player 0 summary functions, and obtain an equivalent flat game $\mathcal{G}^{\mathrm{s}}_{\mathcal{S},\mathcal{A}}$.

Given a regular connectivity tree $\mathcal{T} = \langle T, V \rangle$, that induces a hierarchical system $\mathcal{K}$, we prove Lemma 1 by showing that the flat membership game $\mathcal{G}^{\mathrm{s}}_{\mathcal{S},\mathcal{A}_\varphi}$, where $\mathcal{S}$ is a hierarchical structure whose unwinding is the computation tree $\mathcal{T}_{\mathcal{K},\rho}$, is equivalent to the flat membership game $\mathcal{G}_{K^{\mathcal{T}},\mathcal{A}^{\mathcal{T}}_\varphi}$, of $\mathcal{A}^{\mathcal{T}}_\varphi$ and a Kripke structure $K^{\mathcal{T}}$ whose unwinding is $\mathcal{T}$. Thus, $\mathcal{A}_\varphi$ accepts $\mathcal{T}_{\mathcal{K},\rho}$ iff $\mathcal{A}^{\mathcal{T}}_\varphi$ accepts $\mathcal{T}$. The equivalence of these two games follows from the fact that they have isomorphic arenas and winning conditions. Consequently, our proof of Lemma 1 is mainly syntactic in nature, and amounts to little more then constructing the structures $\mathcal{S}$ and $K^{\mathcal{T}}$, constructing the game $\mathcal{G}_{\mathcal{S},\mathcal{A}_\varphi}$, simplifying it to get $\mathcal{G}^{\mathrm{s}}_{\mathcal{S},\mathcal{A}_\varphi}$, and constructing the membership game $\mathcal{G}_{K^{\mathcal{T}},\mathcal{A}^{\mathcal{T}}_\varphi}$. The remaining technical details can be found in Appendix B. □

We now state our main theorem.

Theorem 2. *The $\langle \mathcal{L}, el \rangle$-synthesis problem is* EXPTIME*-complete for a μ-calculus formula φ, and is* 2EXPTIME*-complete for an* LTL *formula (for el that is at most polynomial in $|\varphi|$ for μ-calculus, or at most exponential in $|\varphi|$ for* LTL*).*

Proof. The lower bounds follow from the same bounds for the classical synthesis problem of flat systems [15, 25], and the fact that it is immediately reducible to our problem if $\mathcal{L}$ contains all the atomic transducers. For the upper bounds, since an APT accepts some tree iff it accepts some regular tree (and $\mathcal{A}^{\mathcal{T}}_\varphi$ obviously only accepts trees which are connectivity trees), by Lemma 1 and Theorem 1, we get that an LTL or a μ-calculus formula φ is $\langle \mathcal{L}, el \rangle$-realizable iff $L(\mathcal{A}^{\mathcal{T}}_\varphi) \neq \emptyset$. Checking the emptiness of $\mathcal{A}^{\mathcal{T}}_\varphi$ can be done either directly, or by first translating it to an equivalent NPT $\mathcal{A}'^{\mathcal{T}}_\varphi$. For reasons that will become apparent in subsection 4.2, we choose the latter. Note that the known algorithms for checking the emptiness of an NPT are such that if $L(\mathcal{A}^{\mathcal{T}}_\varphi) \neq \emptyset$, then one can extract a regular tree in $L(\mathcal{A}^{\mathcal{T}}_\varphi)$ from the emptiness checking algorithm [24]. The upper bounds follow from the analysis given below of the time required to construct $\mathcal{A}^{\mathcal{T}}_\varphi$ and check for its non-emptiness.

By Theorem 1, the number of states $|Q_\varphi|$ and the index k of $\mathcal{A}_\varphi$ is $|Q_\varphi| = 2^{O(|\varphi|)}$, $k = 2$ for LTL, and $|Q_\varphi| = O(|\varphi|)$, $k = O(|\varphi|)$ for μ-calculus. The most time consuming part in the construction of $\mathcal{A}^{\mathcal{T}}_\varphi$ is calculating for every $(\mathcal{K}^E, \sigma, q) \in (\mathcal{L}^{el} \times \Sigma_I \times Q_\varphi)$, the set $Rel(\mathcal{K}^E, \sigma, q)$. Calculating $Rel(\mathcal{K}^E, \sigma, q)$ can be done by checking for every summary function $g \in Sf(\mathcal{K}^E, \sigma, q)$ if it is relevant. Our proof of Lemma 1 also yields that, by [6], the latter can be done in time $O((|K| \cdot |Q_\varphi|)^k \cdot (k+1)^{|E| \cdot |Q_\varphi| \cdot k})$. Observe that the set $Sf(\mathcal{K}^E, \sigma, q)$ is of size $(k+1)^{|E|}$, and that the number of transducers in $\mathcal{L}^{el}$ is $O(\lambda \cdot m^{el})$, where m is the maximal size of any $\mathcal{K} \in \mathcal{L}$. It follows that for an LTL (resp. μ-calculus) formula φ, the automaton $\mathcal{A}^{\mathcal{T}}_\varphi$ can be built in time at most polynomial in the size of the library, exponential in el, and double exponential (resp. exponential) in $|\varphi|$.

We now analyze the time it takes to check for the non-emptiness of $\mathcal{A}^{\mathcal{T}}_\varphi$. Recall that for every $\eta \in (\mathcal{L}^{el} \times \Sigma_I \times Q_\varphi)$, the set $Rel(\eta)$ is of size at most $(k+1)^{el}$, and thus, the size of the transition relation of $\mathcal{A}^{\mathcal{T}}_\varphi$ is polynomial in $|\mathcal{L}|$ and $|\varphi|$, and exponential in el.

Checking the emptiness of $\mathcal{A}_\varphi^{\mathcal{T}}$ is done by first translating it to an equivalent NPT $\mathcal{A}_\varphi^{\prime\mathcal{T}}$. By [20], given an APT with $|Q|$ states and index k, running on Σ-labeled $\mathcal{D}^*$-trees, one can build (in time polynomial in the descriptions of its input and output automata) an equivalent NPT with $(|Q| \cdot k)^{O(|Q| \cdot k)}$ states, an index $O(|Q| \cdot k)$, and a transition relation of size $|\Sigma| \cdot (|Q| \cdot k)^{O(|D| \cdot |Q| \cdot k)}$. It is worth noting that this blow-up in the size of the automaton is independent from the size of the transition relation of $\mathcal{A}_\varphi^{\mathcal{T}}$. By [16, 28], the emptiness of $\mathcal{A}_\varphi^{\prime\mathcal{T}}$ can be checked in time $|\Sigma| \cdot (|Q| \cdot k)^{O(|D| \cdot |Q|^2 \cdot k^2)}$ (and if it is not empty, a witness is returned). Recall that $|\Sigma| = |\mathcal{L}^{el}| = O(\lambda \cdot m^{el})$, and that $|D| = el \cdot |\Sigma_I|$. By substituting the values calculated above for $|Q|$ and k, the theorem follows. □

Note that in Algorithm 1, it is conceivable that the transducer $\mathcal{K}^i$ synthesized at iteration i will be exponential (or even double-exponential for LTL) in the size of the specification formula φ_i. At this point it is probably best to stop the process, refine the specifications, and try again. However, it is important to note that even if the process is continued, and $\mathcal{K}^i$ is added to the library, the time complexity of the succeeding iterations does not deteriorate since the single-round $\langle \mathcal{L}, el \rangle$-synthesis algorithm is only polynomial in the maximal size m of any transducer in the library.

4.2 Enforcing Modularity

In this section, we address two main issues that may hinder the efforts of our single-round $\langle \mathcal{L}, el \rangle$-synthesis algorithm to synthesize a succinct hierarchical transducer $\mathcal{K}$. The first issue is that of ensuring that, when possible, $\mathcal{K}$ indeed makes use of the more complex transducers in the library (especially transducers synthesized in previous rounds) and does not rely too heavily on the less complex, or atomic, transducers. An obvious and most effective solution to this problem is to simply not have some (or all) of the atomic transducers present in the library. The second issue is making sure that $\mathcal{K}$ does not have too many sub-transducers, which can happen if it uses too many copies of the same transducer $\mathcal{K}' \in \mathcal{L}^{=\emptyset}$, each with a different set of exits. We also discuss some other points of interest regarding the synthesis of exits. We address the above issues by constructing, for each constraint we want to enforce on the synthesized transducer $\mathcal{K}$, an APT $\mathcal{A}$, called the constraint monitor, such that $\mathcal{A}$ accepts only connectivity trees that satisfy the constraint. We then synthesize $\mathcal{K}$ by checking the non-emptiness not of $\mathcal{A}_\varphi^{\mathcal{T}}$, but of the product of $\mathcal{A}_\varphi^{\mathcal{T}}$ with all the constraints monitors. Note that a nondeterministic monitor (i.e., an NPT) of exponential size can also be used, without adversely affecting the time-complexity, if the product with it is taken *after* we translate the product of $\mathcal{A}_\varphi^{\mathcal{T}}$ and the other (polynomial) APT monitors, to an equivalent NPT.

A simple and effective way to enforce modularity in Algorithm 1 is that once a transducer $\mathcal{K}^i$ is synthesized in round i, one incorporates in subsequent rounds a monitor that rejects any connectivity tree containing a node labeled by some key sub-transducers of $\mathcal{K}^i$. This effectively enforces any transducer synthesized using a formula that refers to atomic propositions present only in $\mathcal{K}^i$ (and its disallowed sub-transducers) to use $\mathcal{K}^i$, and not try to build its functionality from scratch. As to other ways to enforce modularity, the question of whether one system is more modular than another, or how to construct a modular system, has received many, and often widely different, answers. Here we only discuss how certain simple modularity criteria can be easily implemented on top of our algorithm. For example, some people would argue that a function that has

more than, say, 10 consecutive lines of code in which no other function is called, is not modular enough. A monitor that checks that in no path in a connectivity tree there are more than 10 consecutive nodes labeled with an atomic transducer, can easily enforce such a criterion. We can even divide the transducers in the library into groups, based on how "high level" they are, and enforce lower counts on lower level groups. Essentially, every modularity criterion that can be checked by a polynomial APT, or an exponential NPT, can be used. Enforcing one context-free property can also be done, albeit with an increase in the time complexity. Other non-regular criteria may be enforced by directly modifying the non-emptiness checking algorithm instead of by using a monitor, and we reserve this for future work.

As for the issue of synthesized exits, recall that for each transducer $\mathcal{K}' \in \mathcal{L}^{=\emptyset}$, we can have as many as $\Omega(|\mathcal{K}'|)^{el}$ copies of $\mathcal{K}'$ in $\mathcal{L}^{el}$, each with a different set of exit states. Obviously, we would not like the synthesized transducer $\mathcal{K}$ to use so many copies as sub-transducers. It is not hard to see that one can, for example, build an NPT of size $O(|\mathcal{L}^{el}|)$ that guesses for every $\mathcal{K}' \in \mathcal{L}^{=\emptyset}$ a single set of exits E, and accepts a connectivity tree iff the labels of all the nodes in the tree agree with the guessed exits. Note that after the end of the current round of synthesis, we may choose to add $\mathcal{K}'^{E}$ to the library (in addition, or instead of $\mathcal{K}'$).

Another point to note about the synthesis of exits is that while a transducer $\mathcal{K}$ surely satisfies the formula φ_i it was synthesized for, $\mathcal{K}^E$ may not. Consider for example a transducer $\mathcal{K}$ which is simply a single state, labeled with p, with a self loop. If we remove the loop and turn this state into an exit, it will no longer satisfy $\varphi_i = p \wedge Xp$ or $\varphi_i = Gp$. Now, depending on one's point of view, this may be either an advantage (more flexibility) or a disadvantage (loss of original intent). We believe that this is mostly an advantage, however, in case it is considered a disadvantage, a few possible solutions come to mind. First, for example if $\varphi_i = Gp$, one may wish for $\mathcal{K}$ to remain without exits and enforce $E = \emptyset$. Another option, for example if $\varphi_i = p \wedge Xp$, is to synthesize in round i a modified formula like $\varphi'_i = p \wedge \neg exit \wedge X(p \wedge exit)$, with the thought of exits in mind. Yet another option is to add, at iterations after i, a monitor that checks that if K^E is the label of a node in the connectivity tree then φ_i is satisfied. The monitor can check that φ_i is satisfied inside K^E, in which case the monitor is a single state automaton, that only accepts if E is such that $K^E \models \varphi_i$ (possibly using semantics over truncated paths [9]); alternatively, the monitor can check that φ_i is satisfied in the currently synthesized connectivity tree, starting from the node labeled by K^E, in which case the monitor is based on $\mathcal{A}^{\mathcal{T}}_{\varphi_i}$.

4.3 Incomplete Information

A natural setting that was considered in the synthesis literature is that of incomplete information [15]. In this setting, in addition to the set of input signals I that the system can read, the environment also has internal signals H that the system cannot read, and one should synthesize a system whose behavior depends only on the readable signals, but satisfies a specification which refers also to the unreadable signals. Thus, the specification is given with respect to the alphabet $\Sigma_I = 2^{I \cup H}$, but the behavior of the system must be the same when reading two letters that differ only in their H components. The main source of difficulty is that a finite automaton cannot decide whether or not a computation tree is of a system that behaves in a way which is consistent with its partial view of the input signals. However, since the automaton at the heart of our algorithm does

not run on computation trees, but rather on connectivity trees, handling of incomplete information comes at no cost at all. All we have to do is to define the connectivity trees to be $\mathcal{L}^{el}$-labeled complete $(\{1,...,el\} \times 2^I)$-trees, instead of $(\{1,...,el\} \times 2^{I \cup H})$-trees to ensure that the synthesized transducer behaves in the same way on input letters that differ only in their hidden components (this of course implies that the expression $\bigoplus_{\sigma' \in \Sigma_I}$ in the transition function of $\mathcal{A}^{\mathcal{T}}_{\varphi}$ becomes $\bigoplus_{\sigma' \in 2^I}$). Thus, our algorithm solves, with the same complexity, also the hierarchical synthesis problem with incomplete information.

5 Discussion

We presented an algorithm for the synthesis of hierarchical systems which takes as input a library of hierarchical transducers and a sequence of specification formulas. Each formula drives the synthesis of a new hierarchical transducer based on the current library, which contains all the transducers synthesized in previous iterations together with the starting library. The main challenge in this approach is to come up with a single-round synthesis algorithm that is able to efficiently synthesize the required transducer at each round. We have provided such an algorithm that works efficiently (i.e., not worst than the corresponding one for flat systems) and uniform (i.e., it can handle different temporal logic specifications, including the modal μ-calculus). In order to ensure that the single-round algorithm makes real use of previously synthesized transducers we have suggested the use of auxiliary automata to enforce modularity criteria. We believe that by decoupling the process of enforcing modularity from the core algorithm for single-round synthesis we gain flexibility that allows one to apply different approaches to enforcing modularity, as well as future optimizations to the core synthesis algorithm.

References

[1] R. Alur,M. Arenas,P. Barceló, K. Etessami,N.Immerman, and L. Libkin, First-Order and Temporal Logics for Nested Words. In Logical Methods in Computer Science, vol. 4, 2008.

[2] R. Alur, M. Benedikt, K. Etessami, P. Godefroid, T. W. Reps, and M. Yannakakis. Analysis of recursive state machines. *ACM Trans. Program. Lang. Syst.*, 27(4):786–818, 2005.

[3] R. Alur, S. Chaudhuri, K. Etessami, and P. Madhusudan. On-the-fly reachability and cycle detection for recursive state machines. In *TACAS'05*, LNCS 3440, pages 61–76, 2005.

[4] R. Alur, S. Chaudhuri, and P. Madhusudan. A fixpoint calculus for local and global program flows, In *POPL'06*, ACM, pages 153–165, 2006.

[5] R. Alur and M. Yannakakis. Model checking of hierarchical state machines. *ACM Trans. Program. Lang. Syst.*, 23(3):273–303, 2001.

[6] B. Aminof, O. Kupferman, and A. Murano. Improved model checking of hierarchical systems. In *VMCAI'10*, LNCS 5944, pages 61-77. Springer, 2010

[7] A. Church. Logic, arithmetics, and automata. In *Proc. International Congress of Mathematicians, 1962*, pages 23–35. institut Mittag-Leffler, 1963.

[8] L. de Alfaro and T. A. Henzinger. Interface-based design. In *Engineering Theories of Software-intensive Systems. NATO Science Series: Mathematics, Physics, and Chemistry*, 195, pages 83–104. Springer, 2005.

[9] C. Eisner, D. Fisman, J. Havlicek, Y. Lustig, A. McIsaac, and D. Van Campenhout. Reasoning with temporal logic on truncated paths. In *CAV'O3*, LNCS 2725:(27–39), 2003.
[10] E.A. Emerson. Temporal and modal logic. In J. Van Leeuwen editor, *Handbook of Theoretical Computer Science*, Vol. B, chap. 16, pages 997–1072. Elsevier, MIT Press, 1990.
[11] E.A. Emerson and C. Jutla. Tree automata, μ-calculus and determinacy. In *FOCS'91*, pages 368–377, 1991.
[12] S. Göller and M. Lohrey. Fixpoint logics on hierarchical structures. In *FSTTCS'05*, LNCS 3821, pages 483–494. Springer, 2005.
[13] D. P. Guelev, M. D. Ryan, and P. Y. Schobbens. Synthesising features by games. *Electr. Notes Theor. Comput. Sci.*, 145:79–93, 2006.
[14] D. Janin and I. Walukiewicz. Automata for the modal μ-calculus and related results. In *MFCS'95*, LNCS 969, pages 552–562. Springer-Verlag, 1995.
[15] O. Kupferman and M.Y. Vardi. μ-calculus synthesis. In *MFCS'00*, LNCS 1893:(497–507).
[16] O. Kupferman, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *J. of the ACM*, 47(2):312–360, 2000.
[17] R. Lanotte, A. Maggiolo-Schettini, and A. Peron. Structural model checking for communicating hierarchical machines. In *MFCS*, pages 525–536, 2004.
[18] Y. Lustig and M. Y. Vardi. Synthesis from component libraries. In *FOSSACS'09*, LNCS 5504, pages 395–409. Springer, 2009.
[19] Y. Lustig and M. Y. Vardi. Synthesis from Recursive-Components libraries. In *GandALF'11*, EPTCS 54, pages 1–16, 2011.
[20] D.E. Muller and P.E. Schupp. Alternating automata on infinite trees. *J. of Theor. Comp. Sc.*, 54:267–276, 1987.
[21] Peter Müller. *Modular specification and verification of object-oriented programs*. Springer-Verlag, 2002.
[22] N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive designs. In *VMCAI'06*, LNCS 3855, pages 364–380. Springer, 2006.
[23] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL'89*, pages 179–190. ACM Press, 1989.
[24] M.O. Rabin. Weakly definable relations and special automata. In *Proc. Symp. Math. Logic and Foundations of Set Theory*, pages 1–23. North Holland, 1970.
[25] R. Rosner. *Modular Synthesis of Reactive Systems*. PhD thesis, Weizmann Institute of Science, 1992.
[26] J. Sifakis. A framework for component-based construction extended abstract. In *SEFM'05*, pages 293–300. IEEE Computer Society, 2005.
[27] S. Bliudze and J. Sifakis. Synthesizing Glue Operators from Glue Constraints for the Construction of Component-Based Systems. In *SC'11*, LNCS 6708, pages 51–67. Springer, 2011.
[28] T. Wilke. Alternating tree automata, parity games, and modal μ-calculus. *Bull. Soc. Math. Belg.*, 8(2), 2001.

A Every $\langle \mathcal{L}, el \rangle$-composed transducer $\mathcal{K}$ induces a regular connectivity tree

Let $\mathcal{K} = \langle \Sigma_I, \Sigma_O, \langle \mathcal{K}_1, ..., \mathcal{K}_n \rangle \rangle$, where $\mathcal{K}_1 = \langle W_1, \mathcal{B}_1, in_1, \tau_1, \delta_1, \Lambda_1 \rangle$. We construct a flat transducer $\mathcal{M} = \langle \{1, ..., el\} \times \Sigma_I, \mathcal{L}^{el}, \langle M, m_0, \emptyset, \delta^{\mathcal{T}}, \Lambda^{\mathcal{T}} \rangle \rangle$ whose computation tree $\mathcal{T}_{\mathcal{M}}$ is the required connectivity tree. The elements of $\mathcal{M}$ are as follows:

- $M = W_1 \cup \mathcal{B}_1$, and $m_0 = in_1$.
- If $w \in W_1$, then for every $\sigma \in \Sigma_I$, we have that $\delta^{\mathcal{T}}(w, (1, \sigma)) = \delta_1(w, \sigma)$, and for every $1 < i \leq el$ we (arbitrarily) let $\delta^{\mathcal{T}}(w, (i, \sigma)) = m_0$.
- For $b \in B_1$, let $\mathcal{K}^E \in \mathcal{L}^{el}$ be the sub-transducer that b refers to. For every $\sigma \in \Sigma_I$, if $1 \leq i \leq |E|$ then $\delta^{\mathcal{T}}(b, (i, \sigma)) = \delta_1((b, e), \sigma)$, where $e \in E$ is such that $idx(e, E) = i$; and if $|E| < i \leq el$ then we (arbitrarily) let $\delta^{\mathcal{T}}(b, (i, \sigma)) = m_0$.
- For $w \in W_1$ we have that $\Lambda^{\mathcal{T}}(w) = K_{\varsigma}$, where $\Lambda_1(w) = \varsigma$.
- For $b \in \mathcal{B}_1$ we have that $\Lambda^{\mathcal{T}}(b) = \mathcal{K}_{\tau_1(b)}$.

Note that a son $y \cdot (i, \sigma)$, of a node y in a connectivity tree $\mathcal{T} = \langle T, V \rangle$, is meaningless if i is larger than the number of exits of the transducer $V(y)$. Hence, our choice to direct the corresponding transitions of M to the node m_0, is arbitrary, and was done only for technical completeness.

B Proof of Lemma 1

We begin by introducing the necessary definitions and constructs mentioned in the proof sketch. For more information see [6].

B.1 Hierarchical Structures.

A *hierarchical structure* [5] can be viewed as a hierarchical transducer with an empty input alphabet. Furthermore, unlike a hierarchical transducer which is deterministic, a hierarchical structure has a nondeterministic transition relation. This also allows us to have internal moves from exit nodes. Formally, a hierarchical structure is a tuple $\mathcal{S} = \langle \Sigma_O, \langle \mathcal{S}_1, ..., \mathcal{S}_n \rangle \rangle$, where for every $1 \leq i \leq n$ we have that $\mathcal{S}_i = \langle W_i, \mathcal{B}_i, in_i, Exit_i, \tau_i, \mathcal{R}_i, \Lambda_i \rangle$. The elements $W_i, \mathcal{B}_i, in_i, Exit_i, \tau_i, \Lambda_i$ of the *sub-structure* $\mathcal{S}_i$ are as in a hierarchical transducer, and $\mathcal{R}_i \subseteq (\bigcup_{b \in \mathcal{B}_i} (\{b\} \times Exit_{\tau_i(b)}) \cup W_i) \times (W_i \cup \mathcal{B}_i)$ is a nondeterministic transition relation. As for transducers, edges entering a box implicitly lead to the initial state of the sub-structure it refers to. The special case of a hierarchical structure with a single sub-structure with no boxes and no exits is simply the classical *Kripke structure*, and we denote it by $\mathcal{S} = \langle \Sigma_O, W, in, \mathcal{R}, \Lambda \rangle$.

The definition of a flat expansion $\mathcal{S}^{\mathrm{f}} = \langle \Sigma_O, W^{\mathrm{f}}, in, \mathcal{R}^{\mathrm{f}}, \Lambda^{\mathrm{f}} \rangle$, of a hierarchical structure $\mathcal{S}$, can be obtained by the natural modifications to the definition of the flat expansion of a transducer (see also [6]). Observe that the flat expansion $\mathcal{S}^{\mathrm{f}}$ of a hierarchical structure $\mathcal{S}$ is a Kripke structure, which can be unwound into a tree $\mathcal{T}_{\mathcal{S}} = \langle T_{\mathcal{S}}, V_{\mathcal{S}} \rangle$. We call $\mathcal{T}_{\mathcal{S}}$ the *unwinding* of $\mathcal{S}$. Formally, $\mathcal{T}_{\mathcal{S}}$ is a Σ_O-labeled W^{f}-tree, where a node y in the tree has a son $y \cdot d'$ for every d' for which there is a transition $((last(y), d') \in \mathcal{R}^{\mathrm{f}}$. The label of a node $y \neq \varepsilon$ is $V_{\mathcal{S}}(y) = \Lambda^{\mathrm{f}}(last(y))$, and $V_{\mathcal{S}}(\varepsilon) = \Lambda^{\mathrm{f}}(in)$.

B.2 The Product of a Transducer and a Kripke Structure

Given a hierarchical transducer $\mathcal{K} = \langle \Sigma_I, \Sigma_O, \langle \mathcal{K}_1, ..., \mathcal{K}_n \rangle \rangle$, whose input alphabet Σ_I is the output alphabet of a Kripke structure $\mathcal{S} = \langle \Sigma_I, W, in, \mathcal{R}, \Lambda \rangle$, we can build a hierarchical structure $\mathcal{K} \otimes \mathcal{S}$ by taking the product of $\mathcal{K}$ and $\mathcal{S}$. The hierarchical structure $\mathcal{K} \otimes \mathcal{S}$ has a sub-structure $\mathcal{K}_{i,q}$ for every $2 \leq i \leq n$ and every state $q \in W$, which is essentially the product of the sub-transducer $\mathcal{K}_i$ with $\mathcal{S}$, where the initial state of $\mathcal{K}_i$ is paired with the state q of $\mathcal{S}$. For $i = 1$, we need only the sub-structure $\mathcal{K}_{1,in}$. The hierarchical order of the sub-structures is consistent with the one in $\mathcal{K}$. Thus, the sub-structure $\mathcal{K}_{i,q}$ can be referred to by boxes of a sub-structure $\mathcal{K}_{j,p}$ only if $i > j$. Let $\mathcal{K}_i = \langle W_i, \mathcal{B}_i, in_i, Exit_i, \tau_i, \delta_i, \Lambda_i \rangle$, then $\mathcal{K}_{i,q} = \langle W_i \times W, \mathcal{B}_i \times W, (in_i, q), Exit_i \times W, \tau_{i,q}, \mathcal{R}_{i,q}, \Lambda_{i,q} \rangle$ is such that:

- For $(b,w) \in \mathcal{B}_i \times W$, we have that $\tau_{i,q}(b,w) = (\tau_i(b), w)$.
- For $(u,w) \in W_i \times W$, we have that $((u,w),(v,w')) \in \mathcal{R}_{i,q}$ iff $(w,w') \in \mathcal{R}$ and $\delta_i(u, \Lambda(w')) = v$.
- For $(b,w) \in \mathcal{B}_i \times W$, and an exit $(e,w') \in Exit_{\tau_i(b)} \times W$ of it, we have that $\langle ((b,w),(e,w')),(v,w'') \rangle \in \mathcal{R}_{i,q}$ iff $(w',w'') \in \mathcal{R}$ and $\delta_i((b,e),\Lambda(w'')) = v$.
- For $(u,w) \in W_i \times W$, we have that $\Lambda_{i,q}((u,w)) = (\Lambda_i(u), \Lambda(w))$.

Given $\sigma \in \Sigma_I$, consider the Kripke structure $\mathcal{S}^\sigma = \langle \Sigma_I, \Sigma_I, \sigma, \Sigma_I \times \Sigma_I, \Sigma_I \times \Sigma_I \rangle$, that has one state for every letter in Σ_I (labeled by that letter), its initial state is σ, and it has a transition from every letter to every letter. It is easy to see that we have the following:

Lemma 2. *Given a hierarchical transducer $\mathcal{K}$, and a letter $\sigma \in \Sigma_I$, the computation tree $\mathcal{T}_{\mathcal{K},\sigma}$ can be obtained by unwinding the hierarchical structure $\mathcal{K} \otimes \mathcal{S}^\sigma$.*

B.3 Hierarchical Membership Games

A *hierarchical two-player game* [6] is a game played between two players, referred to as Player 0 and Player 1. The game is defined by means of a hierarchical arena and a winning condition. The players move a token along the hierarchical arena, and the winning condition specifies the objectives of the players as to the sequence of states traversed by the token. A *hierarchical arena* is a hierarchical structure with an empty output alphabet, in which the state space of each of the underlying sub-structures is partitioned into states belonging to Player 0 and states belonging to Player 1. When the token is in a state belonging to one of the players, it chooses a successor to which the token is moved. We refer to the underlying substructures as *sub-arenas*. Formally, a hierarchical two-player game is a pair $\mathcal{G} = (\mathcal{V}, \Gamma)$, where $\mathcal{V} = \langle \mathcal{V}_1, ..., \mathcal{V}_n \rangle$ is a hierarchical arena, and Γ is a winning condition. For every $1 \leq i \leq n$, the sub-arena $\mathcal{V}_i = \langle W_i^0, W_i^1, \mathcal{B}_i, in_i, Exit_i, \tau_i, \mathcal{R}_i \rangle$, is simply a hierarchical structure $\langle \emptyset, \langle W_i^0 \cup W_i^1, \mathcal{B}_i, in_i, Exit_i, \tau_i, \mathcal{R}_i, \emptyset \rangle \rangle$ whose set of states $W_i = W_i^0 \cup W_i^1$ is partitioned to Player 0 states W_i^0, and Player 1 states W_i^1. The parity winning condition $\Gamma : \bigcup_i W_i \to C$ maps all states (of all sub-arenas) to a finite set of colors $C = \{C_{\min}, ..., C_{\max}\} \subset \mathbb{N}$.

Given a hierarchical structure $\mathcal{S} = \langle \Sigma, \langle \mathcal{S}_1, ..., \mathcal{S}_n \rangle \rangle$ and an SAPT $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F \rangle$, the hierarchical two-player game $\mathcal{G}_{\mathcal{S},\mathcal{A}} = (\mathcal{V}, \Gamma)$ for $\mathcal{S}$ and $\mathcal{A}$ is defined as follows. The hierarchical arena $\mathcal{V}$ has a sub-arena $\mathcal{V}_{i,q}$ for every $2 \leq i \leq n$ and state $q \in Q$, which is essentially the product of the structure $\mathcal{S}_i$ with $\mathcal{A}$, where the initial state of $\mathcal{S}_i$ is paired with the state q of $\mathcal{A}$. For $i = 1$, we need only the sub-arena V_{1,q_0}. The hierarchical order

of the sub-arenas is consistent with the one in $\mathcal{S}$. Thus, the sub-arena $\mathcal{V}_{i,q}$ can be referred to by boxes of sub-arena $\mathcal{V}_{j,p}$ only if $i > j$. Let $\mathcal{S}_i = \langle W_i', \mathcal{B}_i', in_i', Exit_i', \tau_i', \mathcal{R}_i', \Lambda_i' \rangle$. Then, the sub-arena $\mathcal{V}_{i,q} = \langle W_{i,q}^0, W_{i,q}^1, \mathcal{B}_{i,q} in_{i,q}, Exit_{i,q}, \tau_{i,q}, \mathcal{R}_{i,q} \rangle$ is defined as follows.

- $W_{i,q}^0 = W_i' \times (Q^\vee \cup Q^{(\varepsilon,\vee)})$, $W_{i,q}^1 = W_i' \times (Q^\wedge \cup Q^{(\varepsilon,\wedge)})$, $in_{i,q} = (in_i', q)$, and $Exit_{i,q} = Exit_i' \times Q^{\vee,\wedge}$.
- $\mathcal{B}_{i,q} = \mathcal{B}_i' \times Q$, and $\tau_{i,q}(b,q) = (\tau_i'(b), q)$.
- For a state $u = (w, \hat{q}) \in W_i' \times Q$, if $\hat{q} \in Q^\varepsilon$ and $\delta(\hat{q}, \Lambda_i'(w)) = \{p_0, ..., p_k\}$, then $(u,v) \in \mathcal{R}_{i,q}$ iff $v \in \{(w,p_0), ..., (w,p_k)\}$; and if $\hat{q} \in Q^{\vee,\wedge}$, then $(u,v) \in \mathcal{R}_{i,q}$ iff $v = (w', \delta(\hat{q}, \Lambda_i'(w)))$ and $(w,w') \in \mathcal{R}_i'$.
- For $(b,p) \in \mathcal{B}_i' \times Q$, and an exit $(e,\hat{q}) \in Exit'_{\tau_i'(b)} \times Q^{\vee,\wedge}$ of this box, we have that $(((b,p),(e,\hat{q})),v) \in \mathcal{R}_{i,q}$ iff $v = (w', \delta(\hat{q}, \Lambda_i'(e)))$ and $((b,e),w') \in \mathcal{R}_i'$.

The winning condition of the game $\mathcal{G}_{\mathcal{S},\mathcal{A}}$ is induced by the acceptance condition of $\mathcal{A}$. Thus, for each state (w,q) of a sub-arena $\mathcal{V}_{i,q}$, we have that $\Gamma(w,q) = F(q)$. For the definition of plays, strategies, etc., please see [6]. It is important to note that, as is the case for flat membership games, a Player 0 strategy for $\mathcal{G}_{\mathcal{S},\mathcal{A}}$ corresponds to a run of $\mathcal{A}$ on the unwinding of $\mathcal{S}$, a memoryless Player 0 strategy corresponds to a memoryless run, and a winning Player 0 strategy corresponds to an accepting run. Furthermore, a Player 0 strategy for a sub-arena $\mathcal{V}_{i,q}$ corresponds to a run of A, starting in state q, on the unwinding of the sub-structure $\mathcal{S}_i$.

Theorem 3. *[6] Given a hierarchical structure $\mathcal{S}$ and a SAPT $\mathcal{A}$, we have that $\mathcal{A}$ accepts the unwinding $\mathcal{T}_\mathcal{S}$ of $\mathcal{S}$, iff Player* 0 *has a winning strategy in the hierarchical game $\mathcal{G}_{\mathcal{S},\mathcal{A}}$.*

We now have all the definitions necessary to build the membership games $\mathcal{G}^{\mathrm{s}}_{\mathcal{K}\otimes\mathcal{S}^\rho,\mathcal{A}_\varphi}$, and $\mathcal{G}_{K^\mathcal{T},\mathcal{A}_\varphi^\mathcal{T}}$.

B.4 The Membership Games $\mathcal{G}^{\mathrm{s}}_{\mathcal{K}\otimes\mathcal{S}^\rho,\mathcal{A}_\varphi}$, and $\mathcal{G}_{K^\mathcal{T},\mathcal{A}_\varphi^\mathcal{T}}$

Given a library $\mathcal{L}$ of hierarchical transducers with input and output alphabets Σ_I and Σ_O, and a bound $el \in \mathbb{N}$, let $\mathcal{T} = \langle T, V \rangle$ be a regular connectivity tree, let $\mathcal{M} = \langle \{1, ..., el\} \times \Sigma_I, \mathcal{L}^{el}, \langle M, m_0, \emptyset, \delta^\mathcal{T}, \Lambda^\mathcal{T} \rangle \rangle$ be a flat transducer such that $\mathcal{T}$ is equal to the (lean) computation tree $\mathcal{T}_\mathcal{M}$, and let $\mathcal{K} = \langle \Sigma_I, \Sigma_O, \langle \mathcal{K}_1, ..., \mathcal{K}_u \rangle \rangle$ be the hierarchical transducer induced by it. Recall that for every $b \in M$, we denote by $E(b)$ the set of top-level exits of $\Lambda^\mathcal{T}(b)$. For the purpose of this proof, it is much more convenient to consider a slightly different version of the induced hierarchical transducer $\mathcal{K}$, where the top level sub-transducer $\mathcal{K}_1$ contains only boxes and no states. That is, we replace every top-level state w, with a box that refers to the atomic transducer K_ς, where ς is such that $\Lambda^\mathcal{T}(w) = \varsigma$. We also relax the definition of hierarchical transducers (as well as hierarchical structures and arenas) to allow the top-level initial state to be not a state but a box. Formally, $\mathcal{K}_1 = \langle \emptyset, M, m_0, \tau_1, \delta_1, \emptyset \rangle$, where:

- For $b \in M$, we have that $\tau_1(b) = i$, where i is such that $\mathcal{K}_i$ is the top-level sub-transducer of $\Lambda^\mathcal{T}(b)$.
- For $b \in M$, we have for every $e \in E(b)$, and every $\sigma \in \Sigma_I$, that $\delta_1((b,e),\sigma) = \delta^\mathcal{T}(b, (idx(e, E(b)), \sigma))$.

It is easy to see that the difference between the version of $\mathcal{K}$ with top-level states, and the modified version without them, is mainly syntactic. Thus, for example, the two versions have isomorphic flat expansions and computation trees, and Lemma 2 also holds if $\mathcal{K}$ has no top-level states. Also, note that since the set of directions of the input trees of an SAPT plays no role in the definition of its run, the computation trees of these two versions of $\mathcal{K}$ are indistinguishable by any SAPT. Finally, one can easily verify that Theorem 3 remains valid also if the hierarchical structure $\mathcal{S}$ has no top-level states.

By Lemma 2, the computation tree $\mathcal{T}_{\mathcal{K},\rho}$ can be obtained by unwinding the hierarchical structure $\mathcal{K} \otimes \mathcal{S}^\rho$. By definition, $\mathcal{K} \otimes \mathcal{S}^\rho$ has a sub-structure $\mathcal{K}_{i,\sigma}$, for every $2 \leq i \leq n$ and every $\sigma \in \Sigma_I$, which is the product of $\mathcal{K}_i$ with $\mathcal{S}^\sigma$, plus a top-level sub-structure $\mathcal{K}_{1,\rho} = \langle \emptyset, M \times \Sigma_I, (m_0,\rho), \emptyset, \tau_{1,\rho}, \mathcal{R}_{1,\rho}, \Lambda_{1,\rho} \rangle$, where:

- For $(b,\sigma) \in M \times \Sigma_I$, we have that $\tau_{1,\rho}(b,\sigma) = (i,\sigma)$, where i is such that $\mathcal{K}_i$ is the top-level sub-transducer of $\Lambda^{\mathcal{T}}(b)$.
- For $(b,\sigma) \in M \times \Sigma_I$, and an exit $(e,\hat{\sigma}) \in \mathit{Exit}_{\tau_1(b)} \times \Sigma_I$ of this box, we have that $\langle ((b,\sigma),(e,\hat{\sigma})),(b',\sigma') \rangle \in \mathcal{R}_{1,\rho}$ iff $\delta^{\mathcal{T}}(b,(\mathit{idx}(e,E(b)),\sigma')) = b'$.

Given a temporal logic formula φ, by Definition 1 and Theorem 1, $\mathcal{K} \models \varphi$ iff $\mathcal{T}_{\mathcal{K},\rho}$ is accepted by the SAPT $\mathcal{A}_\varphi$. Hence, by Theorem 3, $\mathcal{K} \models \varphi$ iff Player 0 has a winning strategy in the hierarchical membership game $\mathcal{G}_{\mathcal{K}\otimes\mathcal{S}^\rho,\mathcal{A}_\varphi}$ of $\mathcal{K} \otimes \mathcal{S}^\rho$ and $\mathcal{A}_\varphi$. Let $\mathcal{A}_\varphi = \langle \Sigma_I \cup \Sigma_O, Q_\varphi, q^0_\varphi, \delta_\varphi, F_\varphi \rangle$ be an SAPT with Q_φ partitioned to $Q_\varphi^{(\varepsilon,\wedge)}$, $Q_\varphi^{(\varepsilon,\vee)}$, $Q_\varphi^\wedge$, and $Q_\varphi^\vee$, and let $\mathcal{K} \otimes \mathcal{S}^\rho$ be as above. By definition, $\mathcal{G}_{\mathcal{K}\otimes\mathcal{S}^\rho,\mathcal{A}_\varphi}$ has a sub-arena $\mathcal{V}_{i,\sigma,q}$ for every $(i,\sigma,q) \in \{1,\ldots,n\} \times \Sigma_I \times Q_\varphi$, which is the product of the sub-structure $\mathcal{K}_i \otimes \mathcal{S}^\sigma$ with $\mathcal{A}^q_\varphi$ (recall that $\mathcal{A}^q_\varphi$ is $\mathcal{A}_\varphi$ with initial state q), plus a top-level sub-arena $V_{1,\rho,q^0_\varphi} = \langle \emptyset, \emptyset, M \times \Sigma_I \times Q_\varphi, (m_0,\rho,q^0_\varphi), \emptyset, \check{\tau}, \check{\mathcal{R}} \rangle$, where:

- For $(b,\sigma,q) \in M \times \Sigma_I \times Q_\varphi$, we have that $\check{\tau}(b,\sigma,q) = (i,\sigma,q)$, where i is such that $\mathcal{K}_i$ is the top-level sub-transducer of $\Lambda^{\mathcal{T}}(b)$.
- For a box $(b,\sigma,q) \in M \times \Sigma_I \times Q_\varphi$, let $\Lambda^{\mathcal{T}}(b) = \mathcal{K}^E$, and let Λ^E be the labeling function of the top-level sub-transducer of $\mathcal{K}^E$. Given an exit $(e,\hat{\sigma},\hat{q}) \in E \times \Sigma_I \times Q_\varphi^{(\varepsilon,\vee)}$ of this box, we have that $\langle ((b,\sigma,q),(e,\hat{\sigma},\hat{q})),(b',\sigma',q') \rangle \in \check{\mathcal{R}}$ iff $q' = \delta_\varphi(\hat{q},(\Lambda^E(e),\hat{\sigma}))$ and $\delta^{\mathcal{T}}(b,(\mathit{idx}(e,E),\sigma')) = b'$.

In [6], in order to solve a hierarchical game, one simplifies it, turning it into an equivalent flat game, by replacing every box of the top-level sub-arena with a gadget that is a 3-level DAG. We briefly recall below the structure of these gadgets, and describe the result of the simplification of the membership game $\mathcal{G}_{\mathcal{K}\otimes\mathcal{S}^\rho,\mathcal{A}_\varphi}$. Let $\beta = (b,\sigma,q)$ be a box of V_{1,ρ,q^0_φ}, let $\mathcal{V}_{i,\sigma,q}$ be the sub-arena that it refers to, let $\Lambda^{\mathcal{T}}(b) = \mathcal{K}^E$, and let $\mathit{Rel}(\mathcal{K}^E,\sigma,q)$ be the set of all relevant summary functions[7] of the runs of $\mathcal{A}^q_\varphi$ on $\mathcal{T}_{\mathcal{K}^E,\sigma}$. A gadget $H_{(\mathcal{K}^E,\sigma,q)}$ contains all the nodes reachable from the root p of the following 3-level DAG structure:

[7] In [6], summary functions were defined in terms of Player 0 strategies. For the special case of the membership game we consider, Player 0 strategies correspond to runs of $\mathcal{A}_\varphi$, and the definition of summary functions given in [6] coincides with the one given in Section 4.1.

- The set of nodes of $H_{(\mathcal{K}^E,\sigma,q)}$ is $\{p\} \cup Rel(\mathcal{K}^E,\sigma,q) \cup (E \times \Sigma_I \times Q_\varphi^{\vee,\wedge} \times C)$. The Player 0 nodes are $\{p\} \cup (E \times \Sigma_I \times Q_\varphi^{\vee} \times C)$, and the Player 1 nodes are $Rel(\mathcal{K}^E,\sigma,q) \cup (E \times \Sigma_I \times Q_\varphi^{\wedge} \times C)$.
- The set of edges is $\bigcup_{g \in Rel(\mathcal{K}^E,\sigma,q)}(\{(p,g)\} \cup \{(g,(h,g(h))) : h \in (E \times \Sigma_I \times Q_\varphi^{\vee,\wedge}) \wedge g(h) \neq \dashv\})$.
- A node $(e,\sigma,q',c) \in (E \times \Sigma_I \times Q_\varphi^{\vee,\wedge} \times C)$ is colored by c. These are the only colored nodes.

The simplification of V_{1,ρ,q_φ^0} is performed by replacing every box $\beta = (b,\sigma,q)$ with a copy of the gadget $H_{(\Lambda^{\mathcal{T}}(b),\sigma,q)}$. To prevent name clashes between copies of the same gadget, we let H^β be a copy of $H_{(\Lambda^{\mathcal{T}}(b),\sigma,q)}$ with all nodes renamed by superscripting them with β. A box β is substituted with H^β by replacing every transition that enters β with a transition that enters the root p^β of H^β, and replacing every transition that exits β through an exit (e,σ,q') with one transition, going out of every leaf of the form (e,σ,q',c) that is present in H^β. Applying this simplification to V_{1,ρ,q_φ^0}, we obtain the flat game $\mathcal{G}^{\mathrm{s}}_{\mathcal{K}\otimes\mathcal{S}^\rho,\mathcal{A}_\varphi} = (V^{\mathrm{s}},\Gamma^{\mathrm{s}})$, where $\mathcal{V}^{\mathrm{s}} = \langle W^{0\mathrm{s}}, W^{1\mathrm{s}}, \emptyset, in^{\mathrm{s}}, \emptyset, \emptyset, \mathcal{R}^{\mathrm{s}}\rangle$, and Γ^{s} are as follows:

- $W^{\mathrm{s}} = \bigcup_{\beta \in (M \times \Sigma_I \times Q_\varphi)}(H^\beta)$, where:
  - $W^{0\mathrm{s}} = \bigcup_{\beta=(b,\sigma,q)\in M\times\Sigma_I\times Q_\varphi}(\{p^\beta\} \cup \{(e,\hat{\sigma},\hat{q},c)^\beta : (e,\hat{\sigma},\hat{q},c) \in (E(b) \times \Sigma_I \times Q_\varphi^{\vee} \times C)\}) \cap W^{\mathrm{s}}$.
  - $W^{1\mathrm{s}} = \bigcup_{\beta=(b,\sigma,q)\in M\times\Sigma_I\times Q_\varphi}(\{g^\beta : g \in Rel(\Lambda^{\mathcal{T}}(b),\sigma,q)\} \cup \{(e,\hat{\sigma},\hat{q},c)^\beta : (e,\hat{\sigma},\hat{q},c) \in (E(b) \times \Sigma_I \times Q_\varphi^{\wedge} \times C)\}) \cap W^{\mathrm{s}}$.
- $in^{\mathrm{s}} = p^{(m_0,\rho,q_\varphi^0)}$.
- For every $\beta = (b,\sigma,q) \in (M \times \Sigma_I \times Q_\varphi)$, with $\Lambda^{\mathcal{T}}(b) = \mathcal{K}^E$, the following transitions are in $\mathcal{R}^{\mathrm{s}}$:
  - $\cup_{g \in Rel(\mathcal{K}^E,\sigma,q)}\{(p^\beta, g^\beta)\}$
  - $\cup_{g \in Rel(\mathcal{K}^E,\sigma,q)}\{(g^\beta,(h,g^\beta(h))) : h \in (E \times \Sigma_I \times Q_\varphi^{\vee,\wedge}) \wedge g^\beta(h) \neq \dashv\}$.
  - Let Λ^E be the labeling function of the top-level sub-transducer of $\mathcal{K}^E$. For every node $(e,\hat{\sigma},\hat{q},c)^\beta$ of H^β, and every $\beta' = (b',\sigma',q') \in (M \times \Sigma_I \times Q_\varphi)$, we have that the transition $((e,\hat{\sigma},\hat{q},c)^\beta, p^{\beta'})$ is in $\mathcal{R}^{\mathrm{s}}$ iff $q' = \delta_\varphi(\hat{q},(\Lambda^E(e),\hat{\sigma}))$ and $b' = \delta^{\mathcal{T}}(b,(idx(e,E),\sigma'))$.
- For every $\beta \in M \times \Sigma_I \times Q_\varphi$, and every node $w = (e,\sigma',q',c)^\beta$ of H^β, we have that $\Gamma^{\mathrm{s}}(w) = c$. All the other nodes[8] are colored by C_{min}.

Theorem 4. *[6] Player* 0 *wins the hierarchical game* $\mathcal{G}_{\mathcal{K}\otimes\mathcal{S}^\rho,\mathcal{A}_\varphi}$ *iff it wins the simplified game* $\mathcal{G}^{\mathrm{s}}_{\mathcal{K}\otimes\mathcal{S}^\rho,\mathcal{A}_\varphi}$.

Applying Theorems 3 and 4 to the constructions above we get that:

Corollary 1. $\mathcal{A}_\varphi$ *accepts* $\mathcal{T}_{\mathcal{K},\rho}$ *iff Player* 0 *wins the game* $\mathcal{G}^{\mathrm{s}}_{\mathcal{K}\otimes\mathcal{S}^\rho,\mathcal{A}_\varphi}$.

[8] In [6], these nodes were left uncolored. However, since there is no cycle that goes only through uncolored nodes, coloring such nodes with C_{min} does not change anything.

We now turn our attention to constructing the membership game $\mathcal{G}_{K^{\mathcal{T}},\mathcal{A}^{\mathcal{T}}_{\varphi}}$. As before, let $\mathcal{M} = \langle \{1,...,el\} \times \Sigma_I, \mathcal{L}^{el}, \langle M, m_0, \emptyset, \delta^{\mathcal{T}}, \Lambda^{\mathcal{T}} \rangle \rangle$ be a flat transducer such that $\mathcal{T}$ is equal to the (lean) computation tree $\mathcal{T}_{\mathcal{M}}$. It is not hard to see that $\mathcal{T}$ can be obtained by unwinding the following Kripke structure[9] $K^{\mathcal{T}} = \langle \mathcal{L}^{el}, W, in, \mathcal{R}, \Lambda \rangle$, where:

- $W = M \times \{1,...,el\} \times \Sigma_I$, and $in = (m_0, 1, \rho)$.
- For $(b,i,\sigma),(b',i',\sigma') \in W$, we have that $((b,i,\sigma),(b',i',\sigma')) \in \mathcal{R}$ iff $\delta^{\mathcal{T}}(b,(i',\sigma')) = b'$.
- For every $(b,i,\sigma) \in W$, we have that $\Lambda(b,i,\sigma) = \Lambda^{\mathcal{T}}(b)$.

Observe that since the transitions of $\mathcal{A}^{\mathcal{T}}_{\varphi}$ mix conjunctions and disjunctions, one cannot construct the arena of the membership game of $K^{\mathcal{T}}$ and $\mathcal{A}^{\mathcal{T}}_{\varphi}$ by directly taking their product. Indeed, doing so would result with nodes of the arena that cannot be assigned to any single player. Hence, we first unfold the transition relation of $\mathcal{A}^{\mathcal{T}}_{\varphi}$, and obtain an equivalent automaton $\tilde{A}^{\mathcal{T}}_{\varphi} = \langle \mathcal{L}^{el}, Q, q_0, \tilde{\delta}, \tilde{F} \rangle$, where the moves from every state are either pure conjunctions or pure disjunctions. Note that this requires that we allow $\tilde{A}^{\mathcal{T}}_{\varphi}$ to have ε-moves. Thus, like an SAPT, the states of $\tilde{A}^{\mathcal{T}}_{\varphi}$ are divided into four sets $Q^{(\varepsilon,\vee)}, Q^{(\varepsilon,\wedge)}, Q^{\vee}$ and $Q^{\wedge}$. However, unlike an SAPT, we allow states in $Q^{\vee,\wedge}$ to send copies in different states to different directions. That is, for $s \in Q^{\vee,\wedge}$, and $\mathcal{K}^E \in \mathcal{L}^{el}$, we have that $\tilde{\delta}(s,\mathcal{K}^E) \subseteq (\{1,...,el\} \times \Sigma_I) \times Q$.

To construct $\tilde{A}^{\mathcal{T}}_{\varphi}$, we simply have to unfold the transition relation of $\mathcal{A}^{\mathcal{T}}_{\varphi}$. That is, for every state $s = (\sigma,q,c) \in \Sigma_I \times Q^{\vee,\wedge}_{\varphi} \times C$ of $\mathcal{A}^{\mathcal{T}}_{\varphi}$, and every $\mathcal{K}^E \in \mathcal{L}^{el}$, we direct the transition from s, when reading $\mathcal{K}^E$, to the entry node of a 3-level DAG gadget $H_{(\mathcal{K}^E,\sigma,q)}$ that unfolds $\delta((\sigma,q,c),\mathcal{K}^E)$. Since $\delta((\sigma,q,c),\mathcal{K}^E)$ is not dependent on the color c, the gadget only depends on σ, q and K^E. We use the same notation for naming these gadgets, as for the gadgets used in the simplification of the game $\mathcal{G}_{\mathcal{K}\otimes\mathcal{S}^{\rho},\mathcal{A}_{\varphi}}$, for the simple reason that they are exactly the same gadgets. In fact, the transition relation of $\mathcal{A}^{\mathcal{T}}_{\varphi}$ is defined the way it is precisely because unfolding it gives these gadgets. Note that resolving the outermost disjunction and conjunction of $\delta((\sigma,q,c),\mathcal{K}^E)$ amounts to choosing some $g \in Rel(\mathcal{K}^E,\sigma,q)$, and some $(e,\sigma',q') \in (E \times \Sigma_I \times Q^{\vee,\wedge}_{\varphi})$, and that once g is chosen, the only leaf of the form $(e,\hat{\sigma},\hat{q},c)$ that is reachable from g is such that $c = g(e,\hat{\sigma},\hat{q})$. Hence, $H_{(\mathcal{K}^E,\sigma,q)}$ faithfully represents the unfolding of $\delta((\sigma,q,c),\mathcal{K}^E)$ even though its leaves include the extra component c. Also, note that since the gadgets are used to simply unfold the transition relation, only the moves going out of their leaves are not ε-moves and correspond to real moves on the input tree. Before we formally describe $\tilde{A}^{\mathcal{T}}_{\varphi}$, observe that even though every gadget $H_{(\mathcal{K}^E,\sigma,q)}$ is used only once in $\tilde{A}^{\mathcal{T}}_{\varphi}$, the names of nodes are not unique across different gadgets. Hence, to get unique names, for every $\eta \in \mathcal{L}^{el} \times \Sigma_I \times Q_{\varphi}$ we subscript the names of nodes in H_{η} with η. Formally, $\tilde{A}^{\mathcal{T}}_{\varphi} = \langle \mathcal{L}^{el}, Q, q_0, \tilde{\delta}, \tilde{F} \rangle$, where:

- $Q = (\Sigma_I \times Q^{\vee,\wedge}_{\varphi} \times C) \cup \{q_0\} \cup \bigcup_{\eta \in (\mathcal{L}^{el} \times \Sigma_I \times Q_{\varphi})} (H_{\eta})$, where:
  - $Q^{(\varepsilon,\vee)} = (\Sigma_I \times Q^{\vee,\wedge}_{\varphi} \times C) \cup \{q_0\} \cup \bigcup_{\eta \in (\mathcal{L}^{el} \times \Sigma_I \times Q_{\varphi})} \{p_{\eta}\}$.

[9] Technically, the unwinding of $K^{\mathcal{T}}$ has the set of directions $M \times \{1,...,el\} \times \Sigma_I$, whereas the set of directions of $\mathcal{T}$ is $\{1,...,el\} \times \Sigma_I$. However, by simply ignoring the M component, we get $\mathcal{T}$.

- $Q^{(\varepsilon,\wedge)} = \bigcup_{\eta\in(\mathcal{L}^{el}\times\Sigma_I\times Q_\varphi)}\{g_\eta : g \in Rel(\eta)\}$.
- $Q^{\vee} = \bigcup_{\eta=(\mathcal{K}^E,\sigma,q)\in(\mathcal{L}^{el}\times\Sigma_I\times Q_\varphi)}\{(e,\hat{\sigma},\hat{q},c)_\eta \in H_\eta : (e,\hat{\sigma},\hat{q},c) \in (E\times\Sigma_I\times Q_\varphi^{\vee}\times C)\}\cap Q$.
- $Q^{\wedge} = \bigcup_{\eta=(\mathcal{K}^E,\sigma,q)\in(\mathcal{L}^{el}\times\Sigma_I\times Q_\varphi)}\{(e,\hat{\sigma},\hat{q},c)_\eta \in H_\eta : (e,\hat{\sigma},\hat{q},c) \in (E\times\Sigma_I\times Q_\varphi^{\wedge}\times C)\}\cap Q$.

– Since certain states are only reachable via ε moves, where the input does not change, $\tilde{\delta}$ is a partial function which is defined only for $s \in Q$, and $\mathcal{K}^E \in \mathcal{L}^{el}$, for which it is possible to reach s when the current input is $\mathcal{K}^E$. Thus:
  - $\tilde{\delta}(q_0,\mathcal{K}^E) = \tilde{\delta}((\rho,q_\varphi^0,C_{min}),\mathcal{K}^E)$.
  - If $s = (\sigma,q,c) \in (\Sigma_I\times Q_\varphi^{\vee,\wedge}\times C)$, then $\tilde{\delta}(s,\mathcal{K}^E) = \{p_{(\mathcal{K}^E,\sigma,q)}\}$.
  - If $s = p_\eta$, where $\eta \in (\mathcal{L}^{el}\times\Sigma_I\times Q_\varphi)$, then $\tilde{\delta}(s,\mathcal{K}^E) = \{g_\eta : g \in Rel(\eta)\}$. If $s = g_\eta$, where g is a summary function in $Rel(\mathcal{K}^E,\sigma,q)$, then $\tilde{\delta}(s,\mathcal{K}^E) = \{(h,g(h)) : h \in (E\times\Sigma_I\times Q_\varphi^{\vee,\wedge}) \wedge g(h) \neq \dashv\}$.
  - If $s = (e,\hat{\sigma},\hat{q},c)_\eta$, then $\tilde{\delta}(s,\mathcal{K}^E) = \bigcup_{\sigma'\in\Sigma_I}\{((idx(e,E),\sigma'),(\sigma',\delta_\varphi(\hat{q},(\Lambda^E(e),\hat{\sigma})),c))\}$, where Λ^E is the labeling function of the top-level sub-transducer of $\mathcal{K}^E$.

– Finally, for every $s \in Q$, if $s = (e,\hat{\sigma},\hat{q},c)_\eta \in Q^{\vee,\wedge}$, or $s = (\sigma,q,c) \in (\Sigma_I\times Q_\varphi^{\vee,\wedge}\times C)$, then $\tilde{F}(s) = c$; otherwise, $\tilde{F}(s) = C_{min}$.

Note that since states in $(\Sigma_I\times Q_\varphi^{\vee,\wedge}\times C)\cup\{q_0\}$ (i.e., the original states of $\mathcal{A}_\varphi^{\mathcal{T}}$) have a single successor per input, their classification as $(\varepsilon,\vee)$ states, and not as $(\varepsilon,\wedge)$ states, is arbitrary. Also, note that since Lemma 1 which we are trying to prove only concerns connectivity trees, we do not care how $\tilde{A}_\varphi^{\mathcal{T}}$ behaves on trees where the root is not labeled by an atomic transducer. Going over the definitions of $\mathcal{A}_\varphi^{\mathcal{T}}$ and $\tilde{A}_\varphi^{\mathcal{T}}$, one can easily see that given a regular connectivity tree $\mathcal{T}$, then $\mathcal{A}_\varphi^{\mathcal{T}}$ accepts the unwinding of $K^{\mathcal{T}}$ iff $\tilde{A}_\varphi^{\mathcal{T}}$ does.

Our last step is to construct the arena of the membership game $\mathcal{G}_{K^{\mathcal{T}},\mathcal{A}_\varphi^{\mathcal{T}}}$. Recall that, intuitively, this arena is the product of $K^{\mathcal{T}}$ and $\tilde{A}_\varphi^{\mathcal{T}}$, and that the successors of every node (w,s), where w is a state of $K^{\mathcal{T}}$ and s is a state of $\tilde{A}_\varphi^{\mathcal{T}}$, are pairs (w',s'), where s' is a successor of s that is sent to the successor w' of w when reading the label of w. Note, however, that we can eliminate some redundancies, as follows. By the last item in the definition of the transition relation $\tilde{\delta}$, for every node $w = (b,i,\sigma)$ of $K^{\mathcal{T}}$ and every state of the form $s = (\sigma',q,c) \in \Sigma_I\times Q_\varphi^{\vee,\wedge}\times C$ of $\tilde{A}_\varphi^{\mathcal{T}}$, the node (w,s) of the product arena is reachable only if $\sigma = \sigma'$; and by the second item in the definition of $\tilde{\delta}$, every such node (w,s) has only a single successor, namely, $(w,p_{(\Lambda^{\mathcal{T}}(b),\sigma,q)})$. Thus, we can simply eliminate the node (w,s) and direct every incoming transition directly to its single successor. Note that since all of the predecessors of (w,s) have the same color c as (w,s), skipping it does not affect the winning condition. For similar reasons, the initial node $((m_0,1,\rho),q_0)$ of the arena can be replaced by its successor $((m_0,1,\rho),p_{(\Lambda^{\mathcal{T}}(m_0),\rho,q_\varphi^0)})$. Formally, after removing the above redundancies, we have that $\mathcal{G}_{K^{\mathcal{T}},\mathcal{A}_\varphi^{\mathcal{T}}} = (\tilde{V},\tilde{\Gamma})$, with $\tilde{\mathcal{V}} = \langle\tilde{W}^0,\tilde{W}^1,\emptyset,\tilde{in},\emptyset,\emptyset,\tilde{\mathcal{R}}\rangle$, where:

– $\tilde{W} = (M\times\{1,...,el\}\times\Sigma_I)\times\bigcup_{\eta\in(\mathcal{L}^{el}\times\Sigma_I\times Q_\varphi)}(H_\eta)$, where:

- $\tilde{W}^0 = (M \times \{1,...,el\} \times \Sigma_I) \times (\bigcup_{\eta=(\mathcal{K}^E,\sigma,q)\in(\mathcal{L}^{el}\times\Sigma_I\times Q_\varphi)}(\{p_\eta\} \cup \{(e,\hat{\sigma},\hat{q},c)_\eta : (e,\hat{\sigma},\hat{q},c) \in (E \times \Sigma_I \times Q_\varphi^\vee \times C)\})) \cap \tilde{W}$.
- $\tilde{W}^1 = (M \times \{1,...,el\} \times \Sigma_I) \times (\bigcup_{\eta=(\mathcal{K}^E,\sigma,q)\in(\mathcal{L}^{el}\times\Sigma_I\times Q_\varphi)}(\{g_\eta : g \in Rel(\eta)\} \cup \{(e,\hat{\sigma},\hat{q},c)_\eta : (e,\hat{\sigma},\hat{q},c) \in (E \times \Sigma_I \times Q_\varphi^\wedge \times C)\})) \cap \tilde{W}$.

– $\tilde{in} = ((m_0,1,\rho), p_{(\Lambda^{\mathcal{T}}(m_0),\rho,q_\varphi^0)})$.

– For every $w = (b,i,\grave{\sigma}) \in M \times \{1,...,el\} \times \Sigma_I$, with $\Lambda^{\mathcal{T}}(b) = \mathcal{K}^E$, and every $\eta = (\mathcal{K}^E,\sigma,q) \in \{\mathcal{K}^E\} \times \Sigma_I \times Q_\varphi$, the following transitions are in $\tilde{\mathcal{R}}$:
  - $\cup_{g\in Rel(\mathcal{K}^E,\sigma,q)}\{\langle (w,p_\eta),(w,g_\eta)\rangle\}$.
  - $\cup_{g\in Rel(\mathcal{K}^E,\sigma,q)}\{\langle (w,g_\eta),(w,(h,g(h)))\rangle : h \in (E \times \Sigma_I \times Q_\varphi^{\vee,\wedge}) \wedge g(h) \neq \dashv\}$
  - Let Λ^E be the labeling function of the top-level sub-transducer of $\mathcal{K}^E$. For every node $(e,\hat{\sigma},\hat{q},c)_\eta$ of H_η, and every $\sigma' \in \Sigma_I$, we have that the transition $\langle (w,(e,\hat{\sigma},\hat{q},c)_\eta),$ $((b',i',\sigma'),p_{\eta'})\rangle$ is in $\tilde{\mathcal{R}}$, iff $b' = \delta^{\mathcal{T}}(b,(idx(e,E),\sigma'))$, and $i' = idx(e,E)$, and $\eta' = (\Lambda^{\mathcal{T}}(b'),\sigma',\delta_\varphi(\hat{q},(\Lambda^E(e),\hat{\sigma})))$.

– Finally, for every $w \in M \times \{1,...,el\} \times \Sigma_I$, for every $\eta \in (\mathcal{L}^{el} \times \Sigma_I \times Q_\varphi)$, and every $s = (e,\sigma',q',c)_\eta$ of H_η, we have that $\tilde{\Gamma}(w,s) = c$. All the other nodes are colored by C_{min}.

Observe that for every $w = (b,i,\grave{\sigma}) \in M \times \{1,...,el\} \times \Sigma_I$, and every $\eta = (\mathcal{K}^E,\sigma,q) \in (\mathcal{L}^{el} \times \Sigma_I \times Q_\varphi)$, by the third item in the definition of $\tilde{\mathcal{R}}$, the only nodes of the form (w,p_η) that have incoming edges are such that $\mathcal{K}^E = \Lambda^{\mathcal{T}}(b)$, and $\grave{\sigma} = \sigma$. By the first and second items in the definition of $\tilde{\mathcal{R}}$ this property propagates, and we get that for every $s_\eta \in H_\eta$ (be it the root, a summary function node, or a leaf) the node (w,s_η) has incoming edges only if $\mathcal{K}^E = \Lambda^{\mathcal{T}}(b)$ and $\grave{\sigma} = \sigma$. We can thus delete all the nodes (w,s_η) in $\tilde{\mathcal{V}}$ for which the above connections between w and η do not exist, since they are not reachable. Also, note that the transitions going out of a node (w,s_η) of $\tilde{\mathcal{V}}$, where $w = (b,i,\sigma) \in M \times \{1,...,el\} \times \Sigma_I$, are completely independent of i, and that the classification of (w,s_η) as a Player 0 or a Player 1 node is also independent of i. Thus, we can safely merge all nodes that differ only in their $\{1,...,el\}$ component into a single node by dropping the $\{1,...,el\}$ component.

After deleting the unreachable nodes mentioned above, and dropping the $\{1,...,el\}$ component, we get that for every $\beta = (b,\sigma,q) \in M \times \Sigma_I \times Q_\varphi$, and every state $s_\eta \in H_{(\Lambda^{\mathcal{T}}(b),\sigma,q)}$, there is exactly one node left in $\tilde{\mathcal{V}}$ that corresponds to β and s_η, that is, the node $((b,\sigma),s_\eta)$. By mapping every such node $((b,\sigma),s_\eta)$ of $\tilde{\mathcal{V}}$, to the node s^β of the arena $\mathcal{V}^s$ of the game $\mathcal{G}^s_{\mathcal{K}\otimes\mathcal{S}^\rho,\mathcal{A}_\varphi}$, we get an isomorphism between (what remained of) $\tilde{\mathcal{V}}$ and $\mathcal{V}^s$. That is, the two arenas (including the coloring) are identical up to the naming of their nodes. Hence, the games $\mathcal{G}_{K^{\mathcal{T}},\mathcal{A}_\varphi^{\mathcal{T}}}$ and $\mathcal{G}^s_{\mathcal{K}\otimes\mathcal{S}^\rho,\mathcal{A}_\varphi}$ are equivalent. Recall that by the construction above Player 0 wins $\mathcal{G}_{K^{\mathcal{T}},\mathcal{A}_\varphi^{\mathcal{T}}}$ iff $\mathcal{A}_\varphi^{\mathcal{T}}$ accepts $\mathcal{T}$, and that by Corollary 1 Player 0 wins $\mathcal{G}^s_{\mathcal{K}\otimes\mathcal{S}^\rho,\mathcal{A}_\varphi}$ iff $\mathcal{A}_\varphi$ accepts $\mathcal{T}_{\mathcal{K},\rho}$, which completes the proof.